MC-202 — Aula 17
Outras árvores: Rubro-Negra, Árvore de Prefixo e Codificação de Huffman

Lehilton Pedrosa

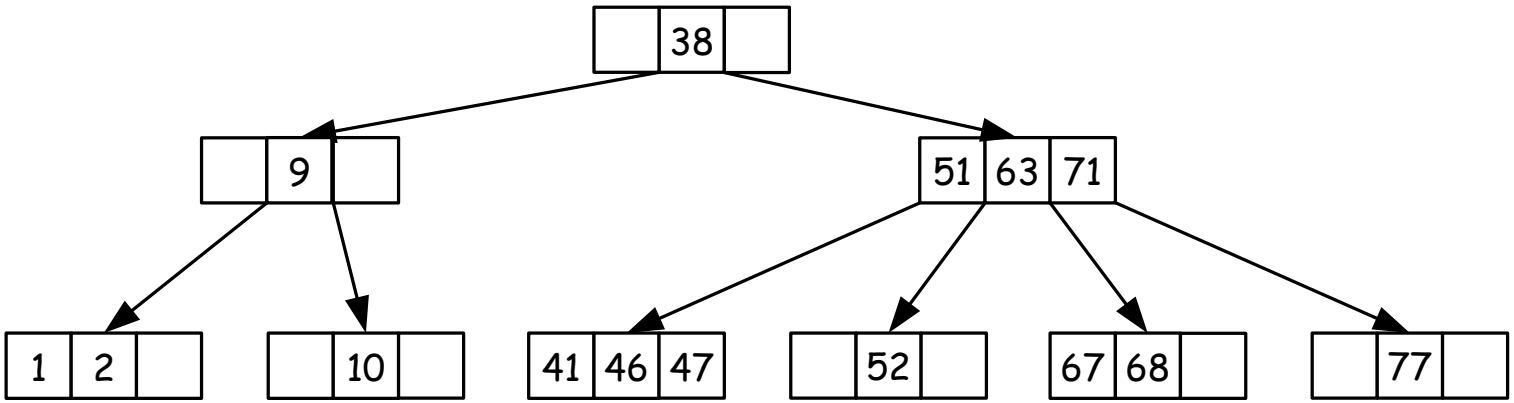
Instituto de Computação – Unicamp

Segundo Semestre de 2015

Roteiro

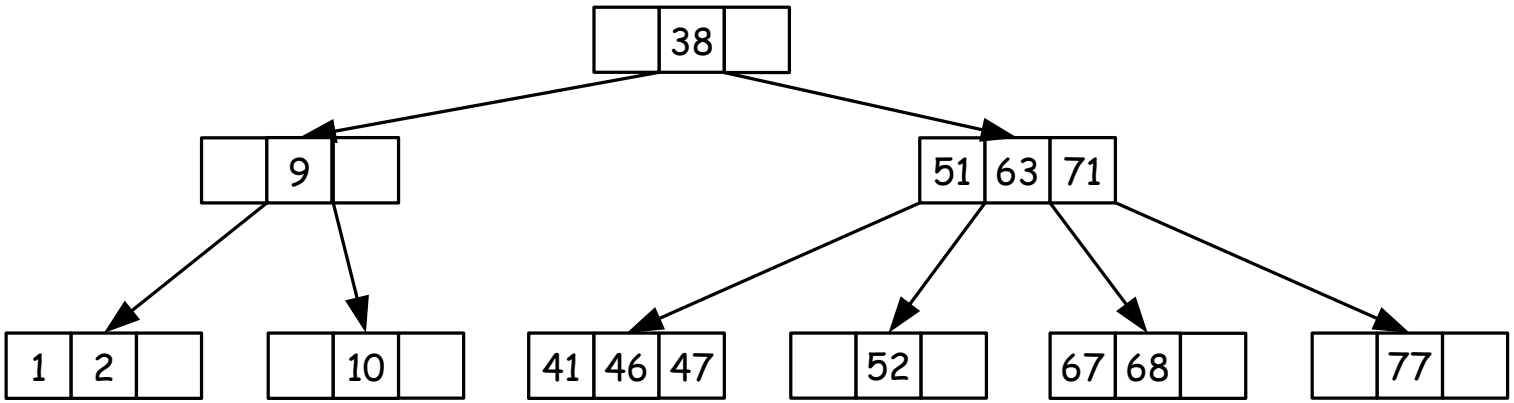
Introdução

Revendo uma Árvore-B de ordem 3.



Introdução

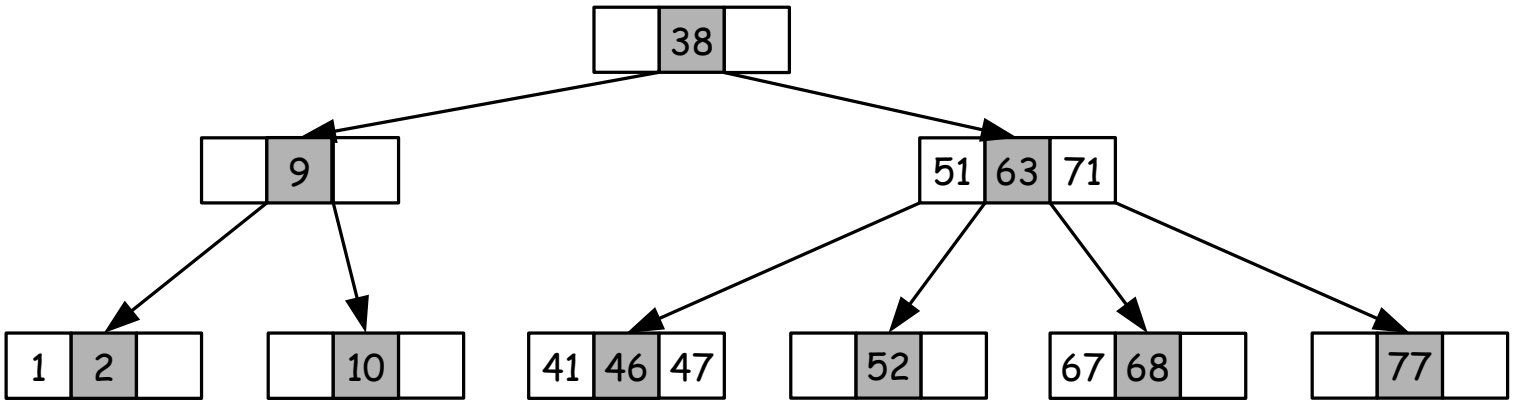
Revendo uma Árvore-B de ordem 3.



Vamos colorir

Introdução

Revendo uma Árvore-B de ordem 3.

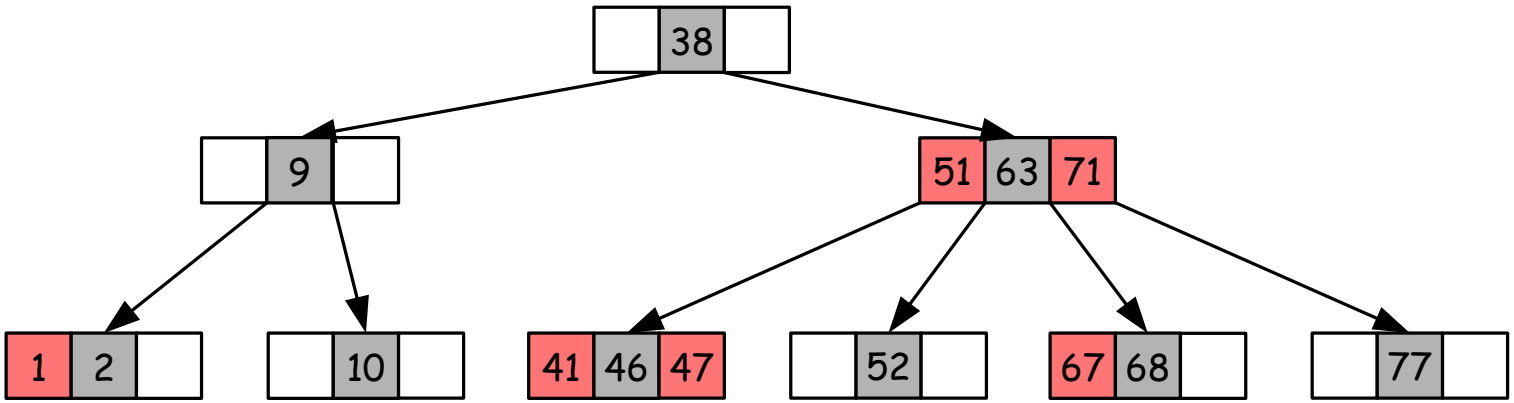


Vamos colorir

- os elementos do meio de **preto**

Introdução

Revendo uma Árvore-B de ordem 3.

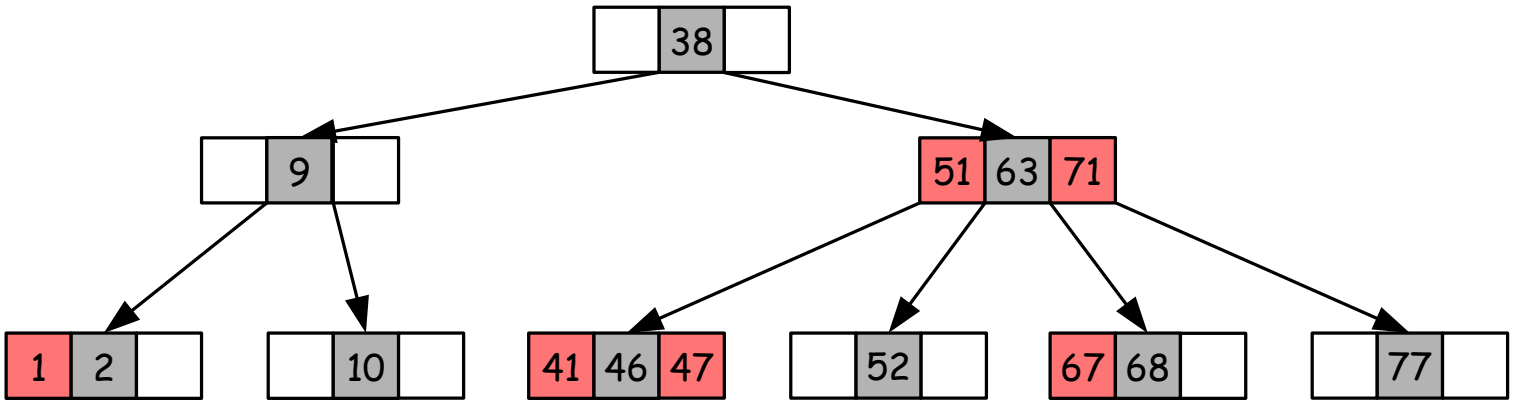


Vamos colorir

- os elementos do meio de **preto**
- os outros elementos de **vermelho**

Introdução

Revendo uma Árvore-B de ordem 3.

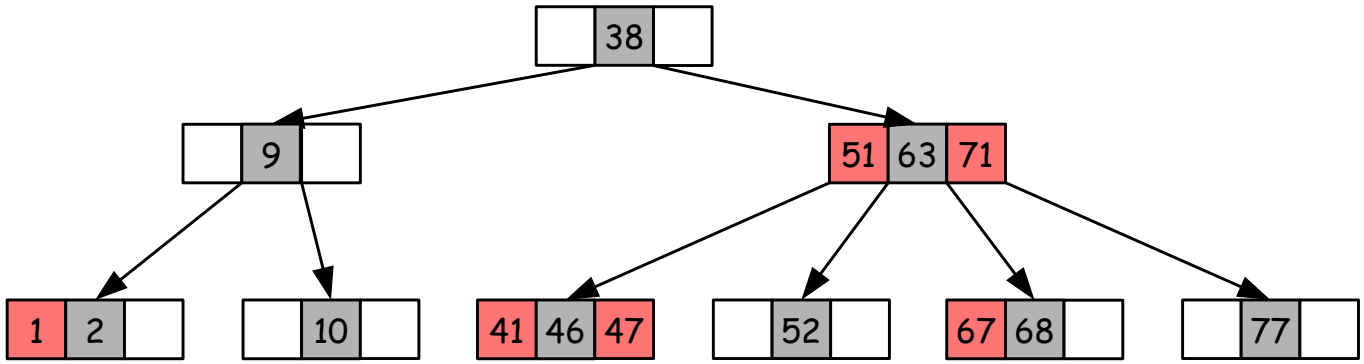


Vamos colorir

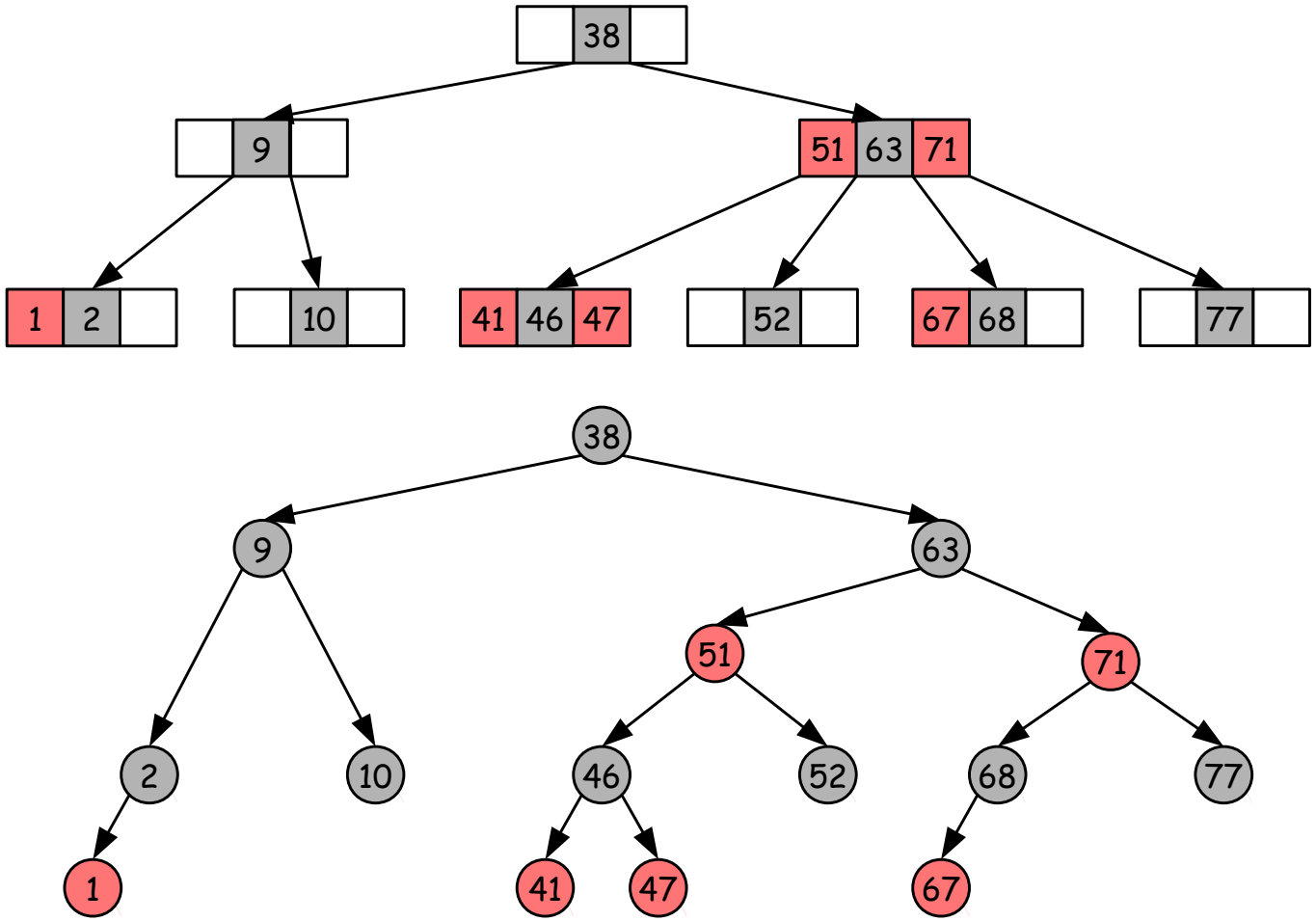
- os elementos do meio de **preto**
- os outros elementos de **vermelho**

Vemos alguma estrutura?

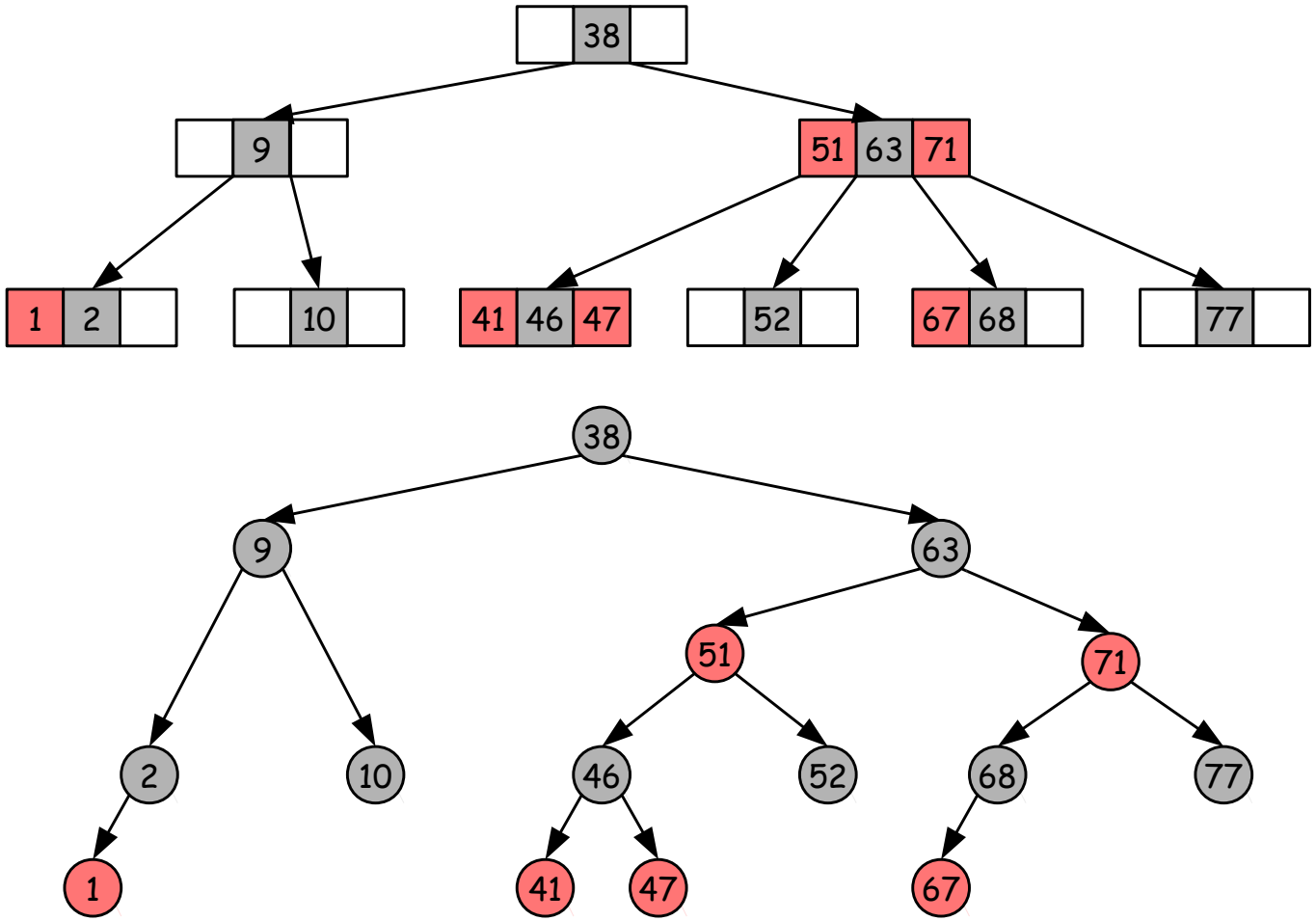
Transformando em árvore binária



Transformando em árvore binária

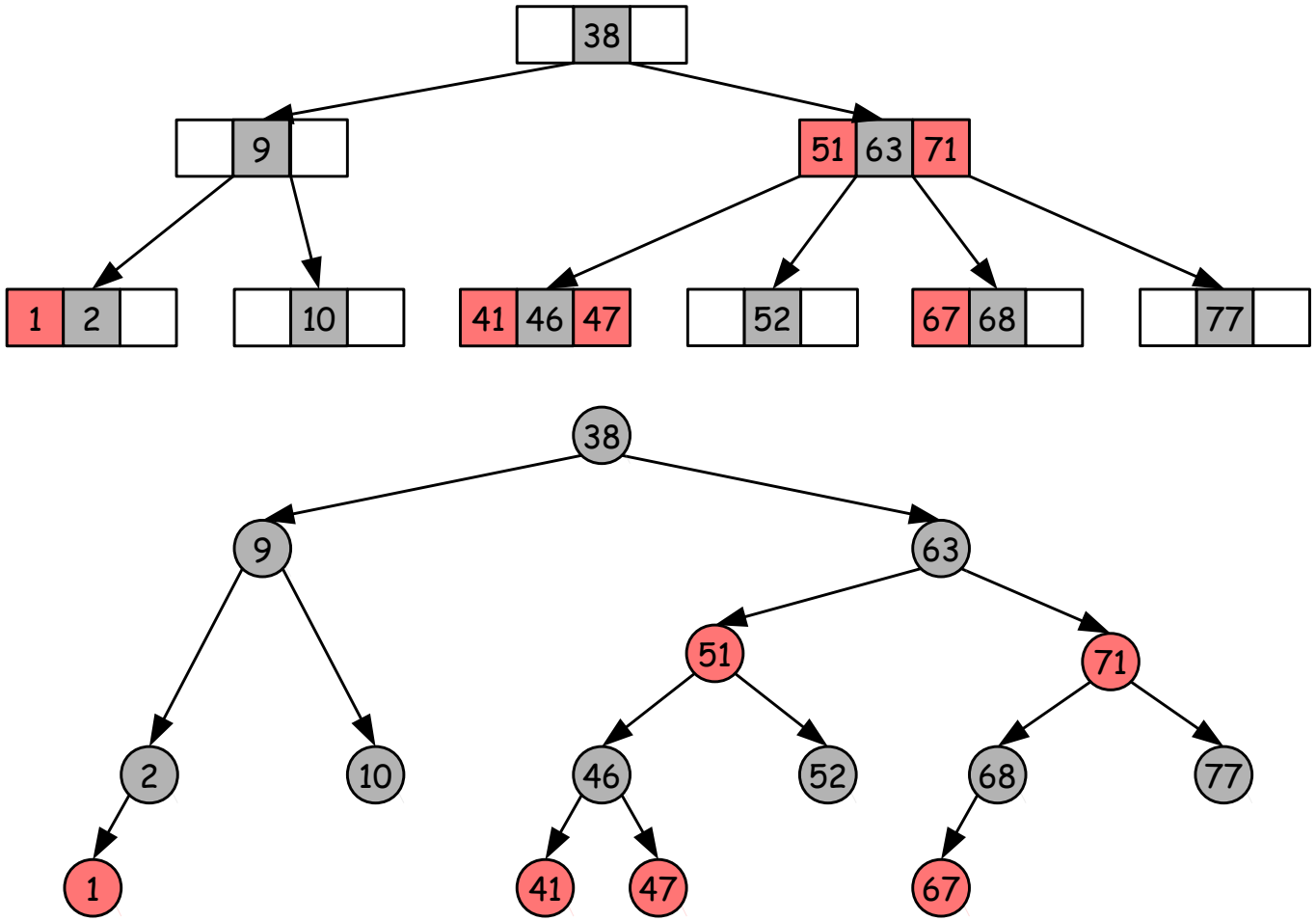


Transformando em árvore binária



Quantos nós pretos há da raiz até uma folha?

Transformando em árvore binária



Quantos nós pretos há da raiz até uma folha? A altura da árvore-B

Árvore Vermelho-Preto

Árvore Vermelho-Preto (*Red-Black Tree*)

- derivadas por Leonidas J. Guibas and Robert Sedgewick (1978)
- uma alternativa às árvores balanceadas AVL
- comumente chamadas de rubro-negras

Árvore Vermelho-Preto

Árvore Vermelho-Preto (*Red-Black Tree*)

- derivadas por Leonidas J. Guibas and Robert Sedgewick (1978)
- uma alternativa às árvores balanceadas AVL
- comumente chamadas de rubro-negras

O número de nós **pretos** da raiz até de uma folha é chamada de altura negra da folha.

Árvore Vermelho-Preto

Árvore Vermelho-Preto (*Red-Black Tree*)

- derivadas por Leonidas J. Guibas and Robert Sedgewick (1978)
- uma alternativa às árvores balanceadas AVL
- comumente chamadas de rubro-negras

O número de nós **pretos** da raiz até de uma folha é chamada de altura negra da folha. Uma árvore vazia é definida como um nó **NIL** de cor preta.

Árvore Vermelho-Preto

Árvore Vermelho-Preto (*Red-Black Tree*)

- derivadas por Leonidas J. Guibas and Robert Sedgewick (1978)
- uma alternativa às árvores balanceadas AVL
- comumente chamadas de rubro-negras

O número de nós **pretos** da raiz até de uma folha é chamada de altura negra da folha. Uma árvore vazia é definida como um nó **NIL** de cor preta.

Definição

Uma árvore de busca é chamada Árvore Vermelho-Preto se ela é vazia ou

1. todo nó é **preto** ou **vermelho**;
2. a raiz é **preta**;
3. os filhos de um nó **vermelho** são **pretos**; e
4. a altura negra de toda folha é a mesma.

Especificação do nó

Nó de árvore Vermelho-Preto

```
typedef struct NoArvRN {
    int chave;
    struct NoArvRN *esq, *dir, *pai;
    enum { PRETO , VERMELHO } cor;
} *NoArvRN;
```

Inserindo

Inserção

1. insere nó normalmente (como em árvore de busca comum)
2. pinta o nó de vermelho
3. corrigimos as propriedades (propagando até a raiz)

Inserindo

Inserção

1. insere nó normalmente (como em árvore de busca comum)
2. pinta o nó de vermelho
3. corrigimos as propriedades (propagando até a raiz)

Quando as propriedades são violadas?

Inserindo

Inserção

1. insere nó normalmente (como em árvore de busca comum)
2. pinta o nó de vermelho
3. corrigimos as propriedades (propagando até a raiz)

Quando as propriedades são violadas?

- quando nó x vermelho for filho de um nó vermelho

Inserindo

Inserção

1. insere nó normalmente (como em árvore de busca comum)
2. pinta o nó de vermelho
3. corrigimos as propriedades (propagando até a raiz)

Quando as propriedades são violadas?

- quando nó x vermelho for filho de um nó vermelho
- quando o nó x vermelho é a raiz

Inserindo

Inserção

1. insere nó normalmente (como em árvore de busca comum)
2. pinta o nó de vermelho
3. corrigimos as propriedades (propagando até a raiz)

Quando as propriedades são violadas?

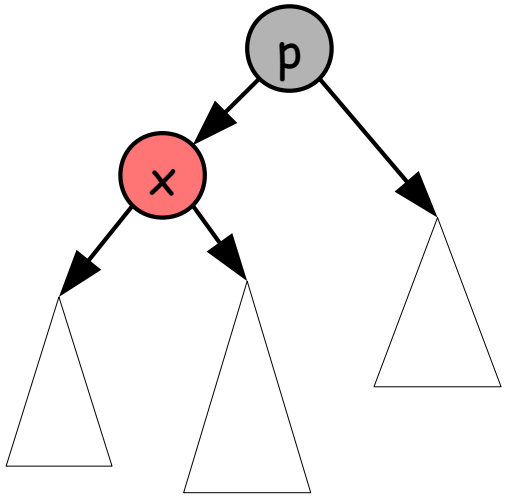
- quando nó x vermelho for filho de um nó vermelho
- quando o nó x vermelho é a raiz

Estratégia de correção

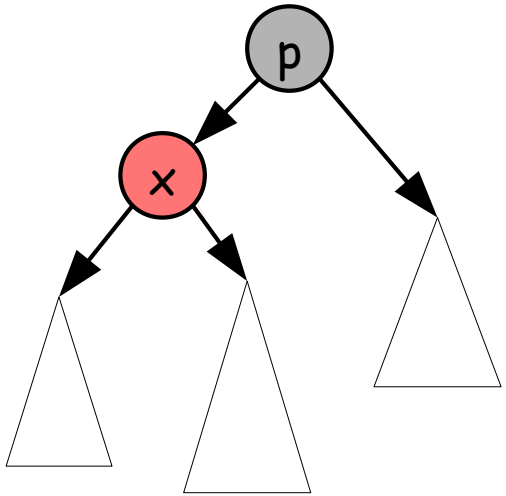
Enquanto algum nó x violar uma propriedade:

- realiza correções
- atualiza o x

Caso 1: pai é preto

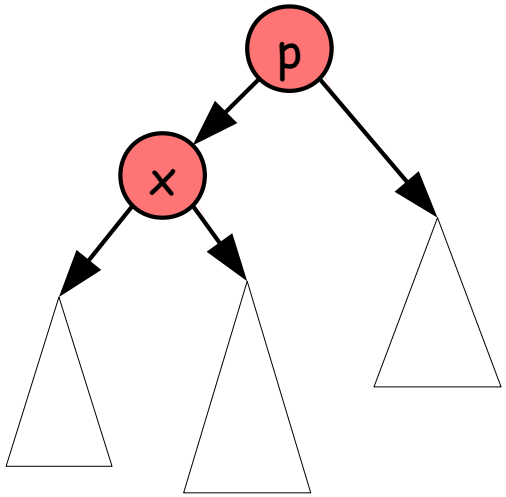


Caso 1: pai é preto

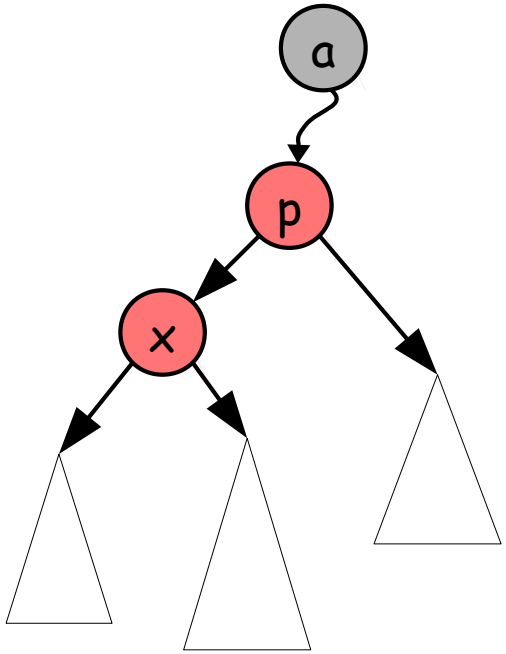


- **Procedimento:** não faz nada
- **Continua:** termina

Caso 2: pai é vermelho

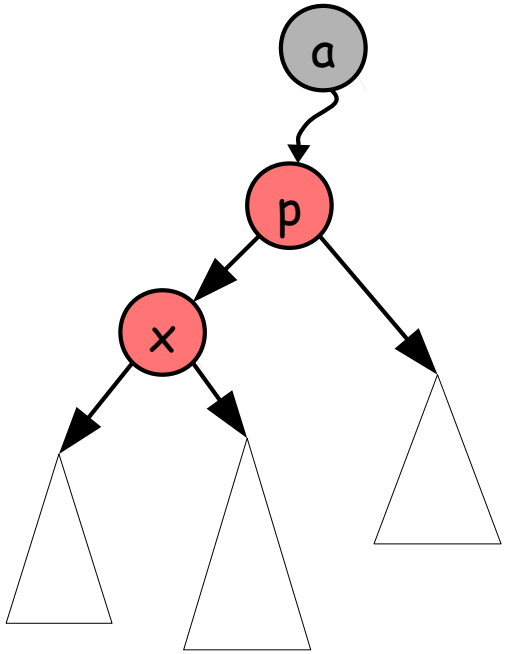


Caso 2: pai é vermelho



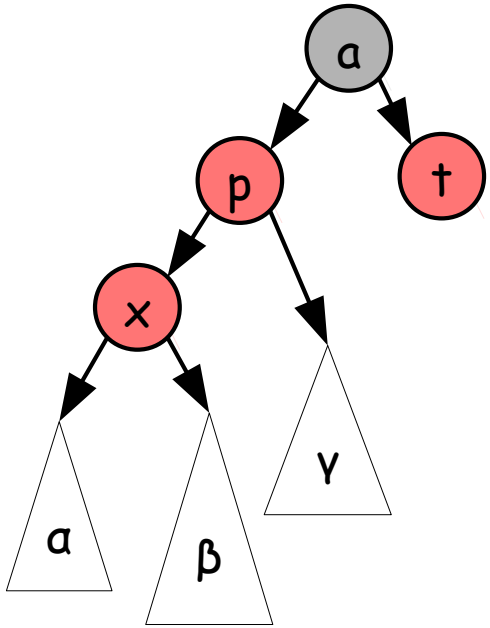
- existe um avô

Caso 2: pai é vermelho

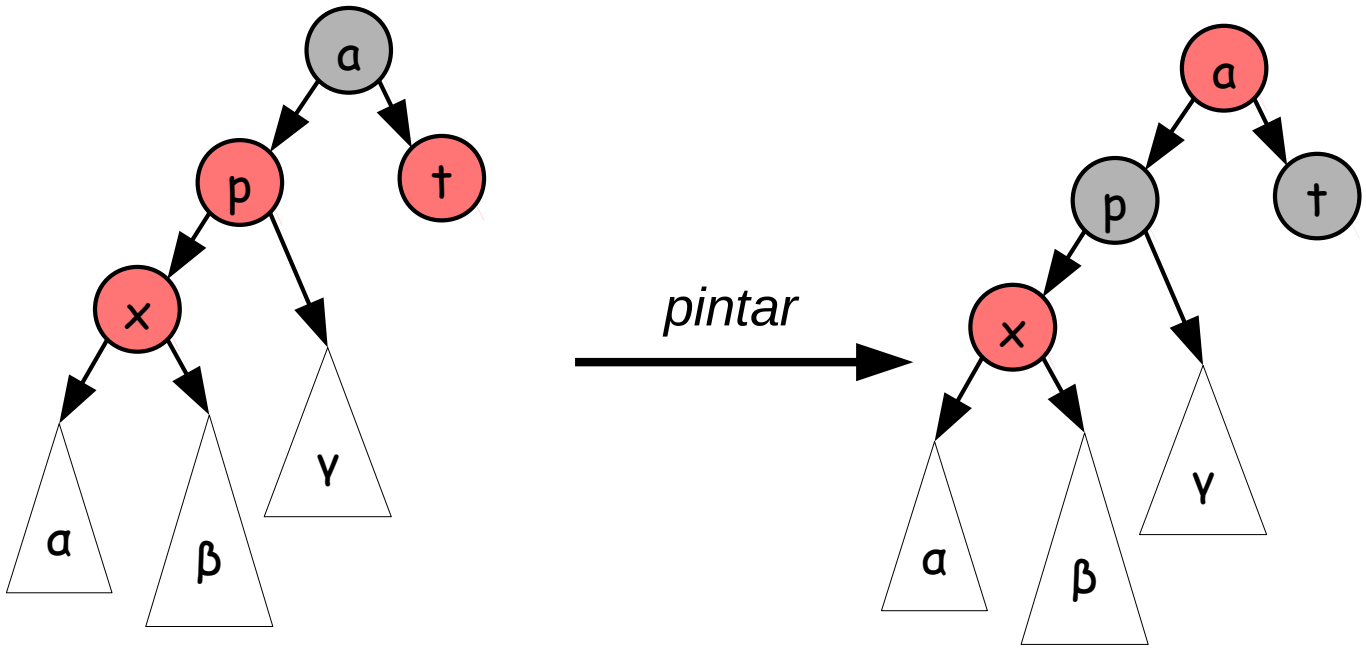


- existe um avô
- vamos olhar para o tio

Caso 2 (a): tio existe e é vermelho

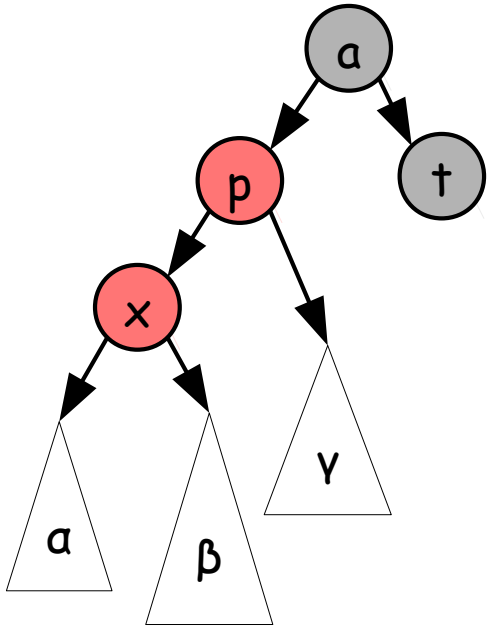


Caso 2 (a): tio existe e é vermelho

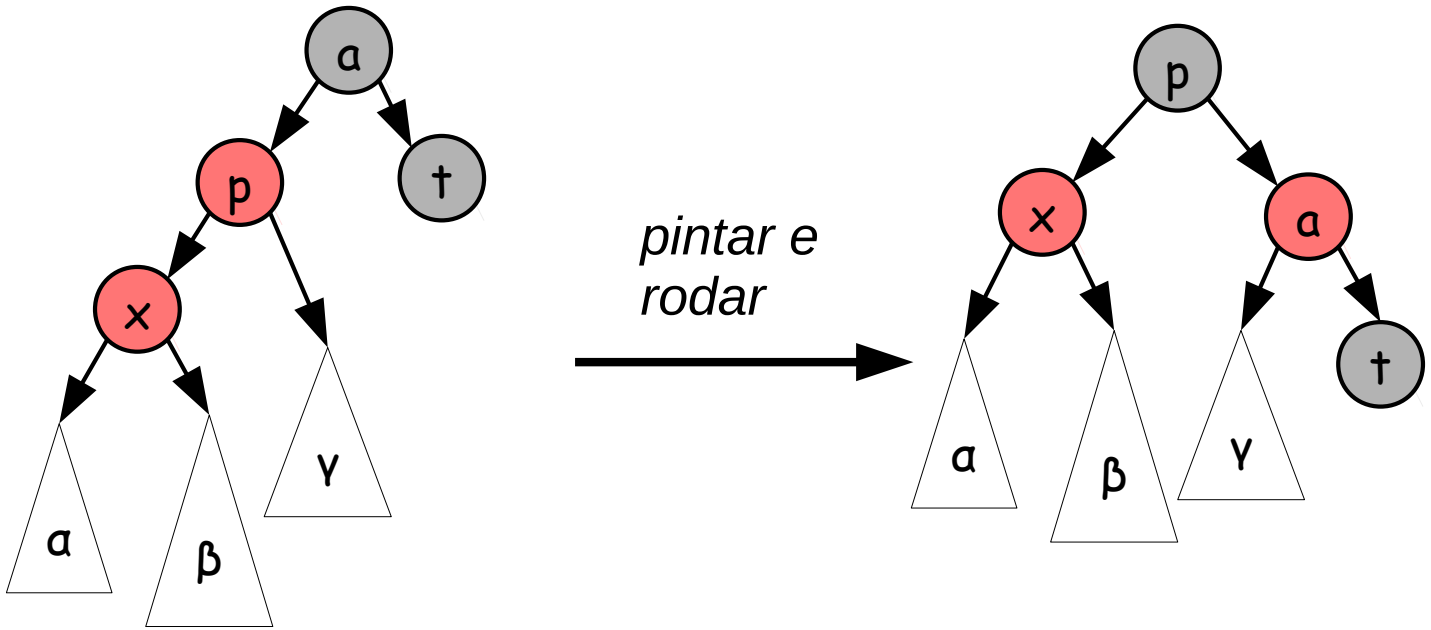


- **Procedimento:** pintamos
- **Continua:** continua com nó *a*

Caso 2 (b): tio é preto e está filho alinhado com o pai

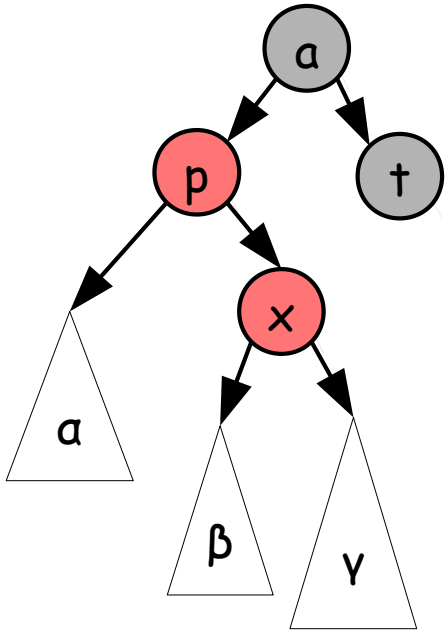


Caso 2 (b): tio é preto e está filho alinhado com o pai

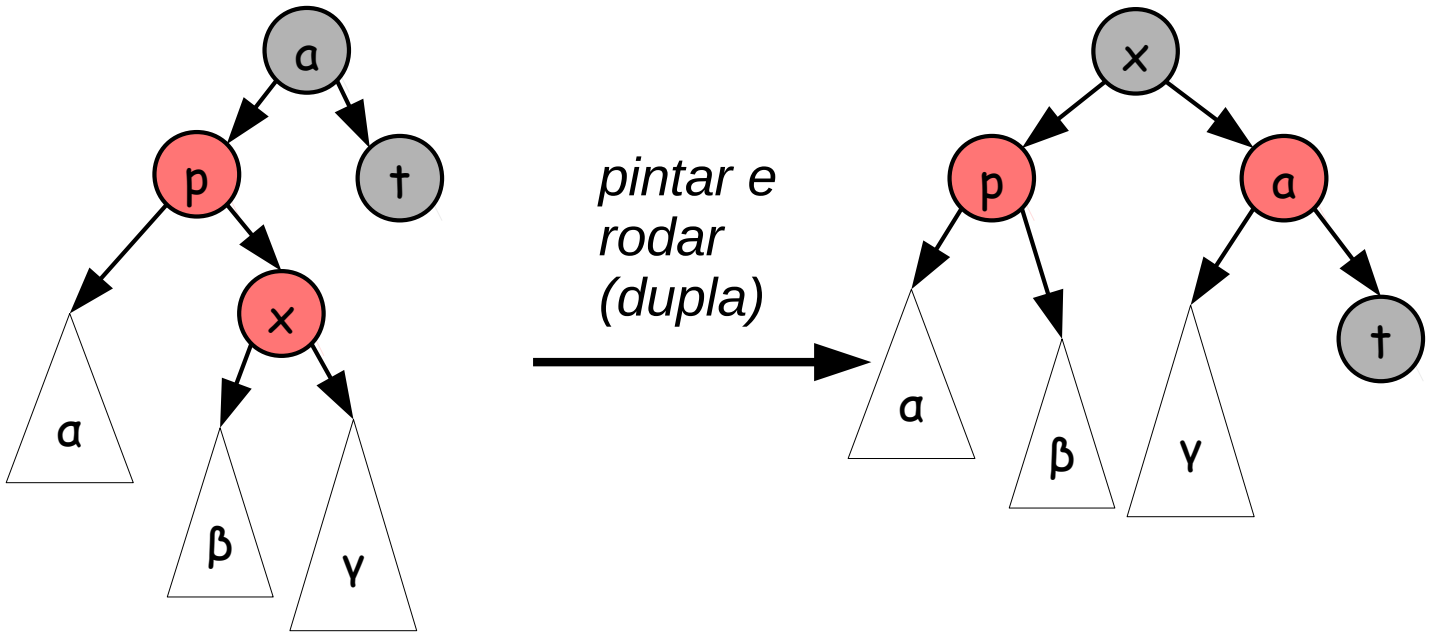


- **Procedimento:** pintamos e rotação simples
- **Continua:** termina

Caso 2 (c): tio é preto e filho está desalinhado com o pai

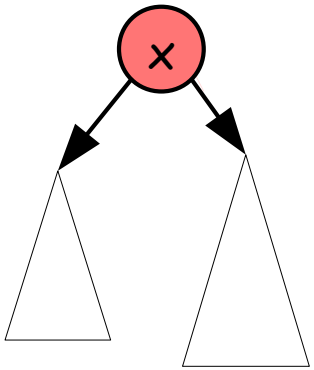


Caso 2 (c): tio é preto e filho está desalinhado com o pai

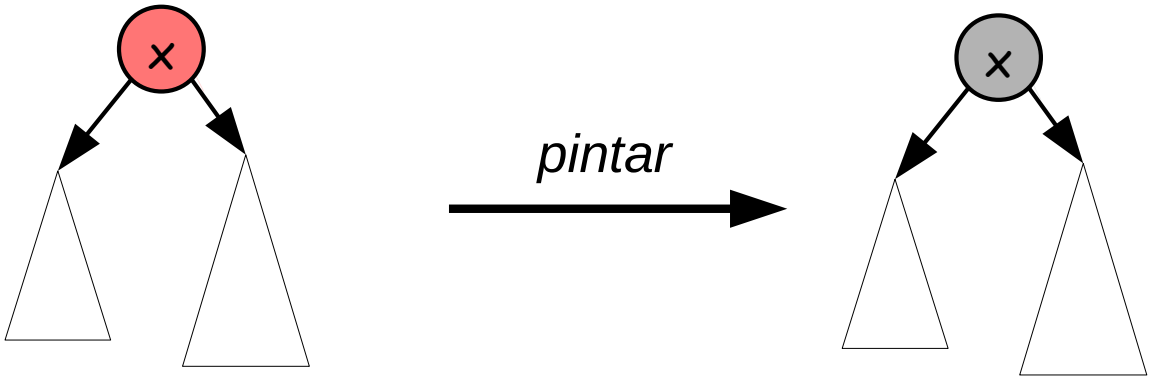


- **Procedimento:** pintamos e rotação dupla
- **Continua:** termina

Caso 3: nó é raiz

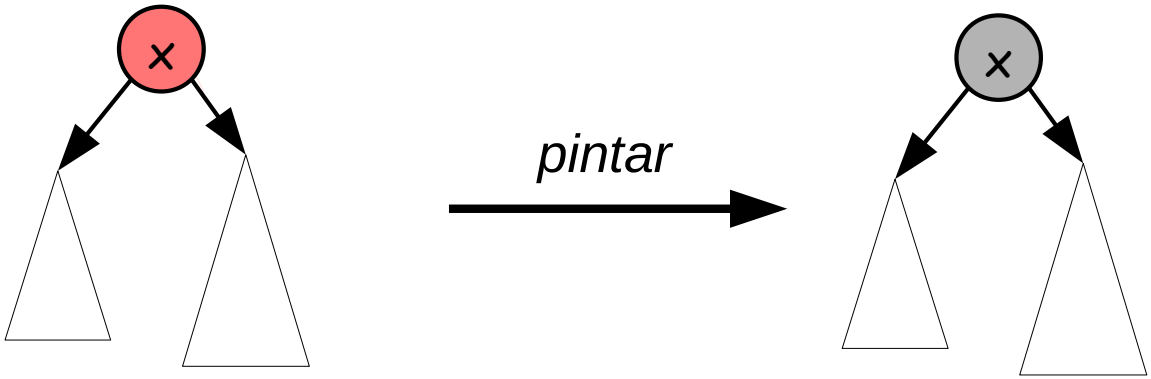


Caso 3: nó é raiz



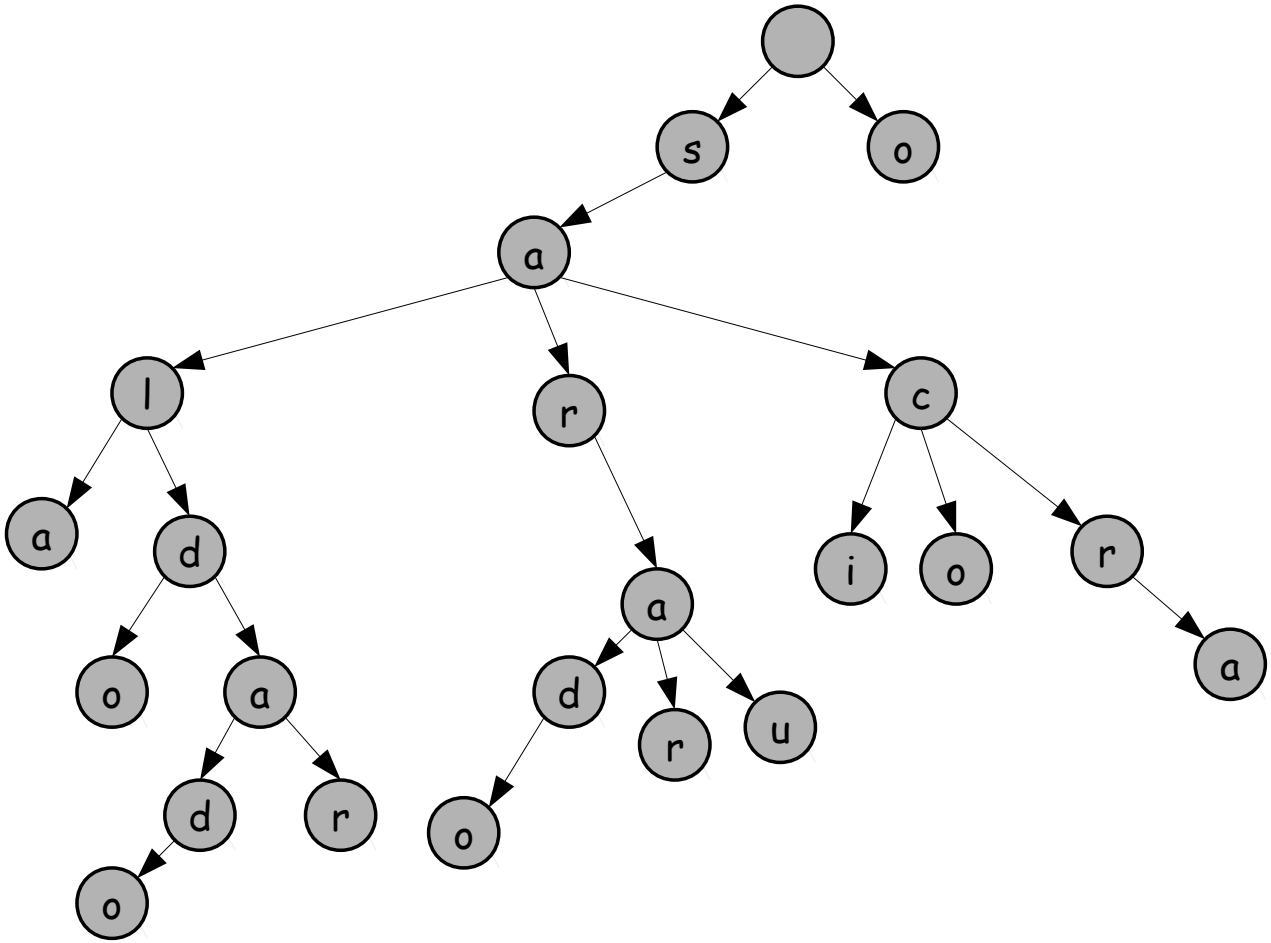
- **Procedimento:** pintamos
- **Continua:** termina

Caso 3: nó é raiz



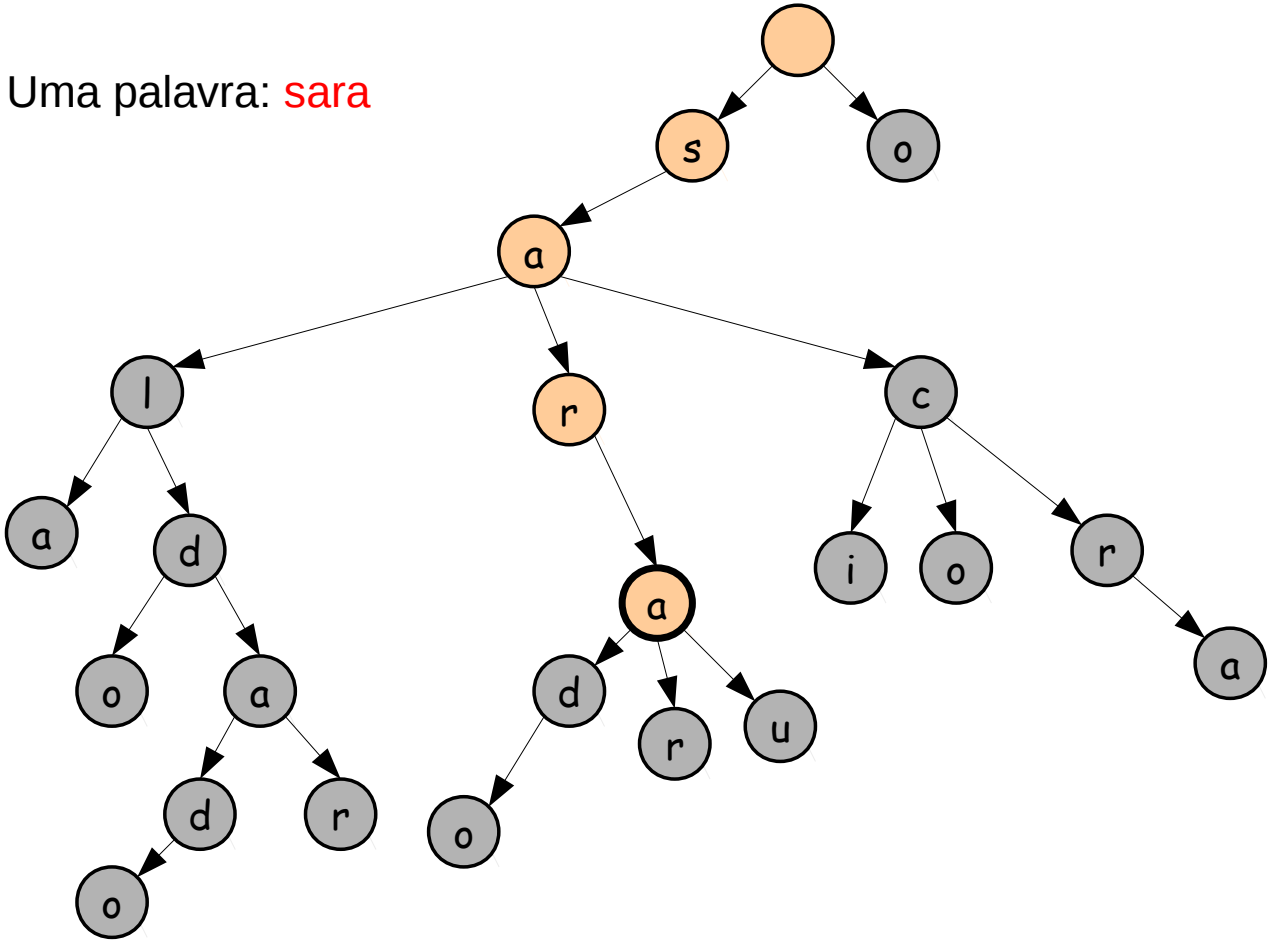
- **Procedimento:** pintamos (aumenta a altura negra!)
- **Continua:** termina

Procurando uma palavra

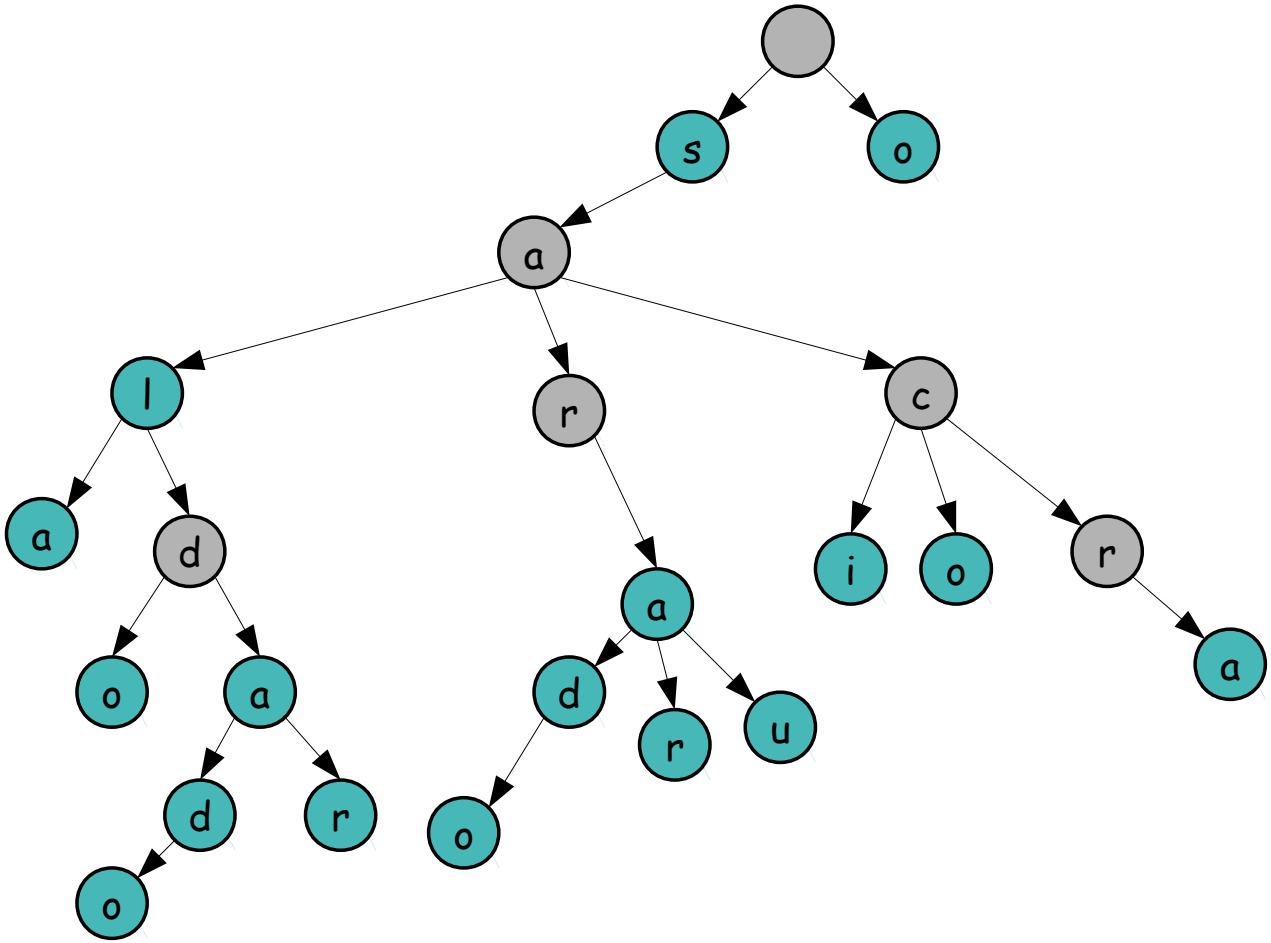


Procurando uma palavra

Uma palavra: sara



Procurando uma palavra



Marcamos os nós que terminam uma palavra.

Árvores de prefixos

Trie

- árvore de busca para chaves do tipo "string"
- busca: só precisamos percorrer a chave
- o nome vem de *retrieval*

Árvores de prefixos

Trie

- árvore de busca para chaves do tipo "string"
- busca: só precisamos percorrer a chave
- o nome vem de *retrieval*

Implementações

- cada nó tem um ponteiro para cada letra do alfabeto (26 ponteiros!)
- cada nó tem uma lista de pares rótulo-ponteiro

Algumas palavras com muitos as

a ala abacatada
da balada abala
a bala aba

Algumas palavras com muitos as

a ala abacatada
da balada abala
a bala aba

Pergunta: como codificar usando a menor quantidade de memória?

Criando uma tabela de símbolos

Temos 7 caracteres:

Criando uma tabela de símbolos

Temos 7 caracteres: no mínimo 3 bits por letra!

Criando uma tabela de símbolos

Temos 7 caracteres: no mínimo 3 bits por letra!

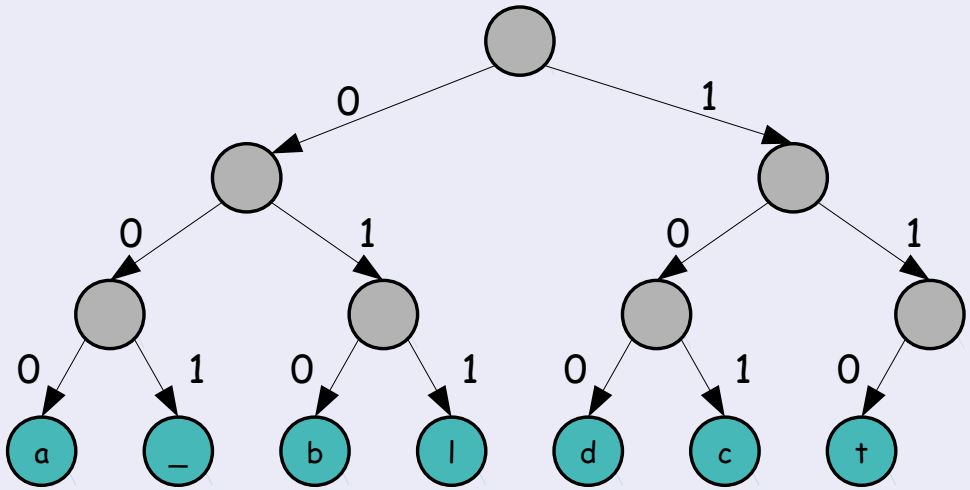
letra	**a**	**_**	**b**	**l**	**d**	**c**	**t**	**total**
frequência	20	8	5	4	3	1	1	42
código	000	001	010	011	100	101	110	126

Criando uma tabela de símbolos

Temos 7 caracteres: no mínimo 3 bits por letra!

letra	**a**	**_**	**b**	**l**	**d**	**c**	**t**	**total**
frequência	20	8	5	4	3	1	1	42
código	000	001	010	011	100	101	110	126

Árvore de símbolos

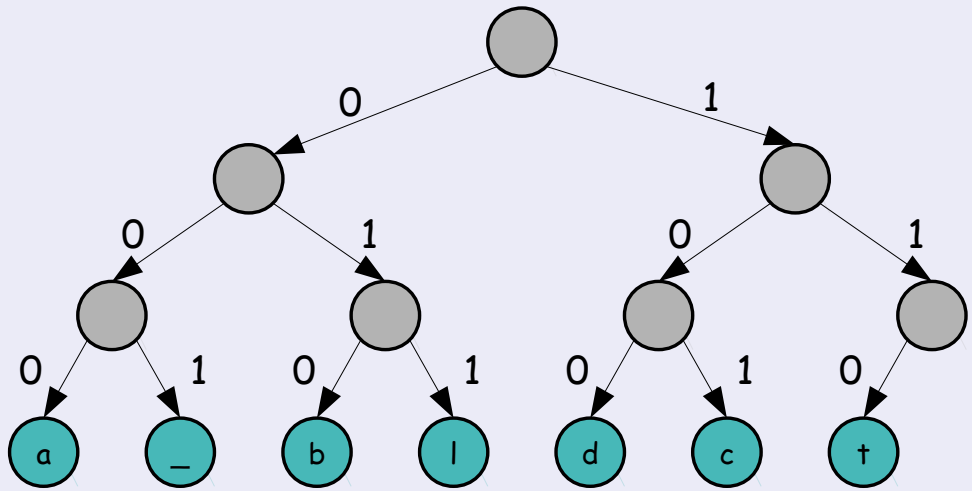


Criando uma tabela de símbolos

Temos 7 caracteres: no mínimo 3 bits por letra!

letra	**a**	**_**	**b**	**l**	**d**	**c**	**t**	**total**
frequência	20	8	5	4	3	1	1	42
código	000	001	010	011	100	101	110	126

Árvore de símbolos



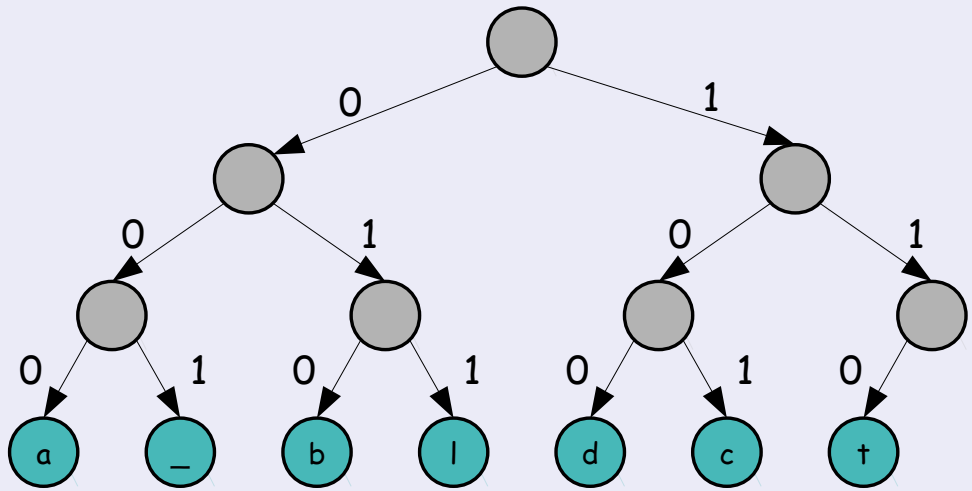
Somente folhas podem ser terminais.

Criando uma tabela de símbolos

Temos 7 caracteres: no mínimo 3 bits por letra!

letra	**a**	**_**	**b**	**l**	**d**	**c**	**t**	**total**
frequência	20	8	5	4	3	1	1	42
código	000	001	010	011	100	101	110	126

Árvore de símbolos



Somente folhas podem ser terminais. Por quê?

Melhorando: Codificação de Huffman

E se pudermos usar códigos de tamanho variável?

Melhorando: Codificação de Huffman

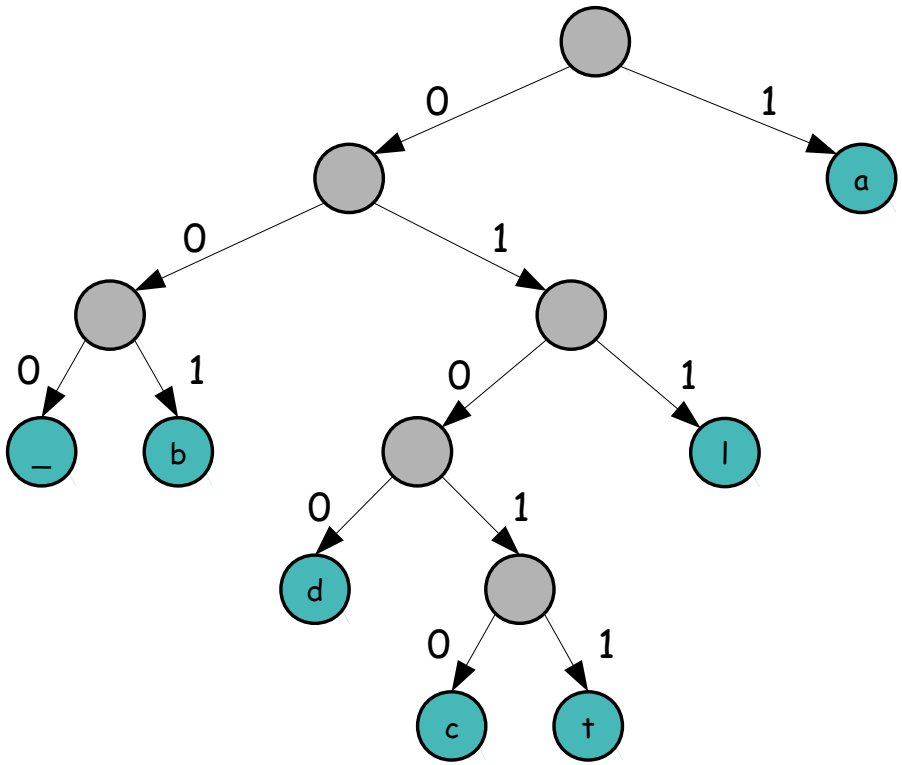
E se pudermos usar códigos de tamanho variável?

letra	**a**	**_**	**b**	**l**	**d**	**c**	**t**	**total**
frequência	20	8	5	4	3	1	1	42
tamanho fixo	000	001	010	011	100	101	110	126

Melhorando: Codificação de Huffman

E se pudermos usar códigos de tamanho variável?

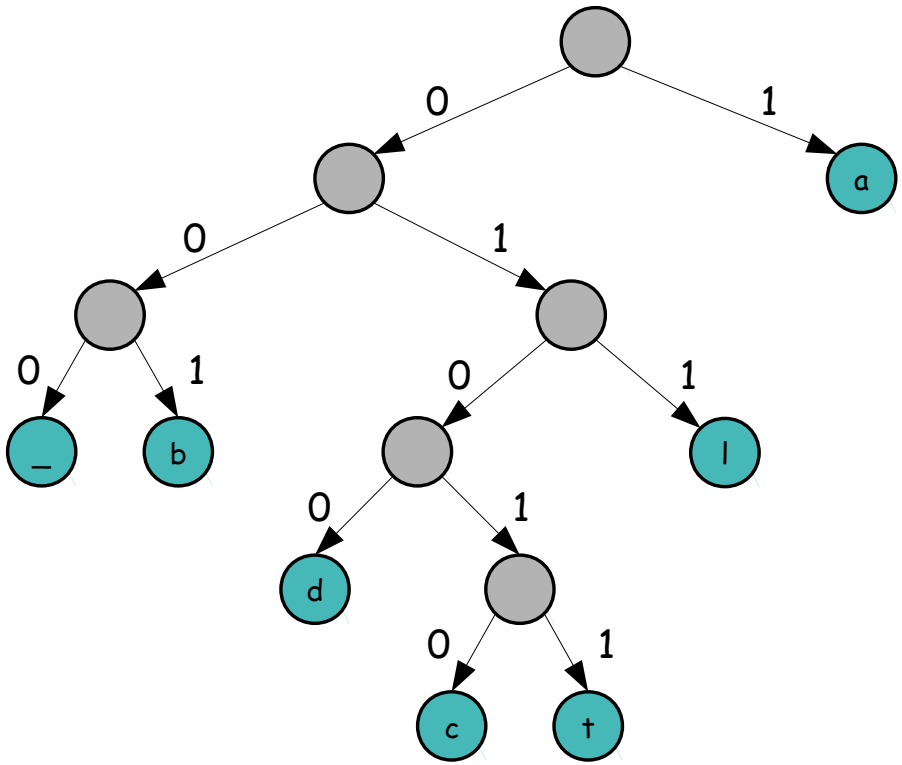
letra	**a**	**_**	**b**	**l**	**d**	**c**	**t**	**total**
frequência	20	8	5	4	3	1	1	42
tamanho fixo	000	001	010	011	100	101	110	126



Melhorando: Codificação de Huffman

E se pudermos usar códigos de tamanho variável?

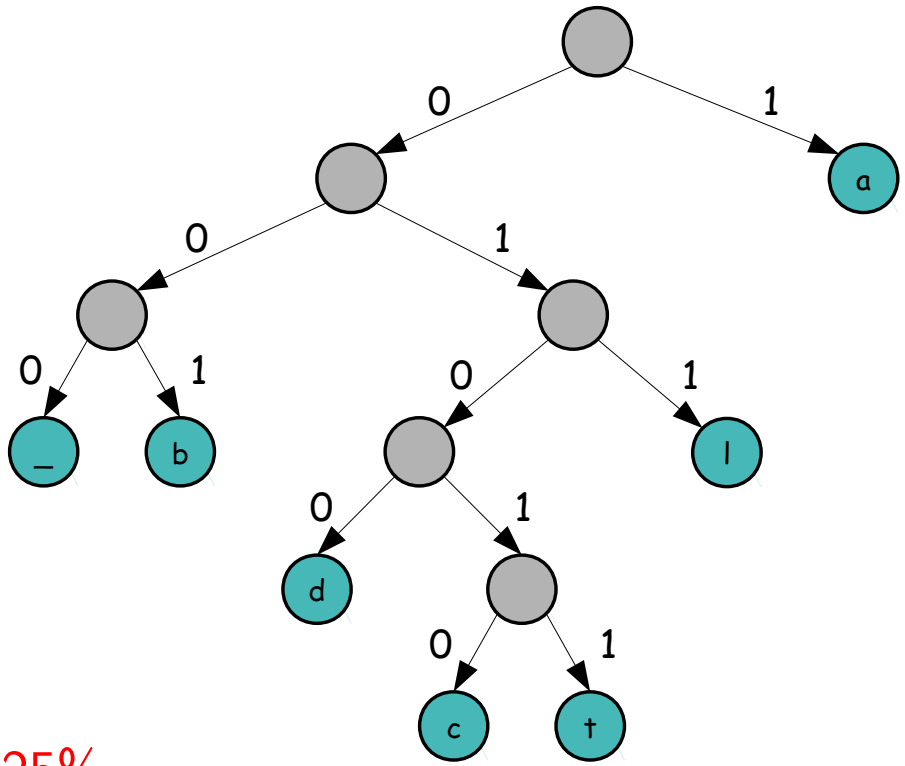
letra	**a**	**_**	**b**	**l**	**d**	**c**	**t**	**total**
frequência	20	8	5	4	3	1	1	42
tamanho fixo	000	001	010	011	100	101	110	126
tamanho variável	1	000	001	011	0100	01010	01011	93



Melhorando: Codificação de Huffman

E se pudermos usar códigos de tamanho variável?

letra	**a**	**_**	**b**	**l**	**d**	**c**	**t**	**total**
frequência	20	8	5	4	3	1	1	42
tamanho fixo	000	001	010	011	100	101	110	126
tamanho variável	1	000	001	011	0100	01010	01011	93



Economizamos $\approx 25\%$.

Árvore de Huffman

Codificação de Huffman

É a codificação que gera a menor sequência de bits de uma sequência de símbolos

Árvore de Huffman

Codificação de Huffman

É a codificação que gera a menor sequência de bits de uma sequência de símbolos

- algoritmo para criar árvore descoberto por David A. Huffman (1952)

Árvore de Huffman

Codificação de Huffman

É a codificação que gera a menor sequência de bits de uma sequência de símbolos

- algoritmo para criar árvore descoberto por David A. Huffman (1952)
- podem existir várias árvores de Huffman para uma sequência

Árvore de Huffman

Codificação de Huffman

É a codificação que gera a menor sequência de bits de uma sequência de símbolos

- algoritmo para criar árvore descoberto por David A. Huffman (1952)
- podem existir várias árvores de Huffman para uma sequência (por quê?)

Árvore de Huffman

Codificação de Huffman

É a codificação que gera a menor sequência de bits de uma sequência de símbolos

- algoritmo para criar árvore descoberto por David A. Huffman (1952)
- podem existir várias árvores de Huffman para uma sequência (por quê?)

Aplicações:

- compressão de dados
- imagem JPEG, MP3 etc.

Árvore de Huffman

Codificação de Huffman

É a codificação que gera a menor sequência de bits de uma sequência de símbolos

- algoritmo para criar árvore descoberto por David A. Huffman (1952)
- podem existir várias árvores de Huffman para uma sequência (por quê?)

Aplicações:

- compressão de dados
- imagem JPEG, MP3 etc.

Implementação:

- Arquivos mantém codificação
- árvore fixa por "idioma".

Árvore de Huffman

Codificação de Huffman

É a codificação que gera a menor sequência de bits de uma sequência de símbolos

- algoritmo para criar árvore descoberto por David A. Huffman (1952)
- podem existir várias árvores de Huffman para uma sequência (por quê?)

Aplicações:

- compressão de dados
- imagem JPEG, MP3 etc.

Implementação:

- Arquivos mantém codificação
- árvore fixa por "idioma".

Variante:

- Codificação adaptativa por Newton Faller (1973)

Criando a árvore de Huffman

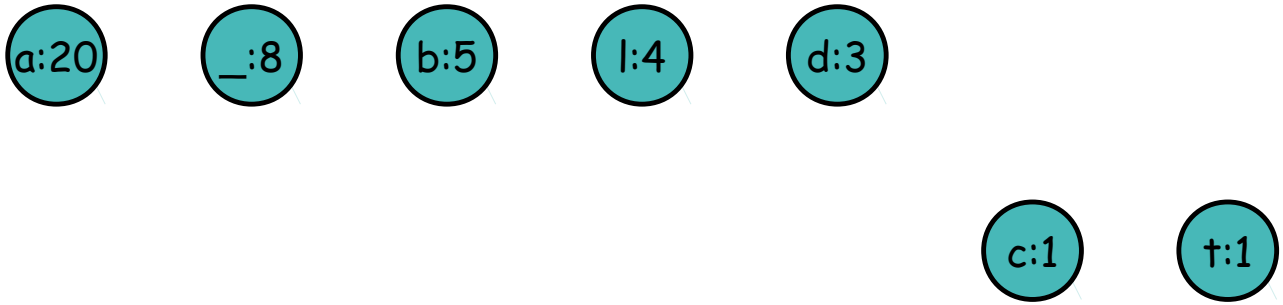
Algoritmo

1. Conte a frequência de cada letra e crie um nó
2. Insira todos nós em uma filha de prioridade (de frequência)
3. Enquanto houver 2 elementos na fila:
   1. Remova os dois elementos a, b com menor prioridade
   2. Crie um nó x com a e b como filhos esquerdo (0) e direito (1)
   3. Insira x na fila com a soma das frequência

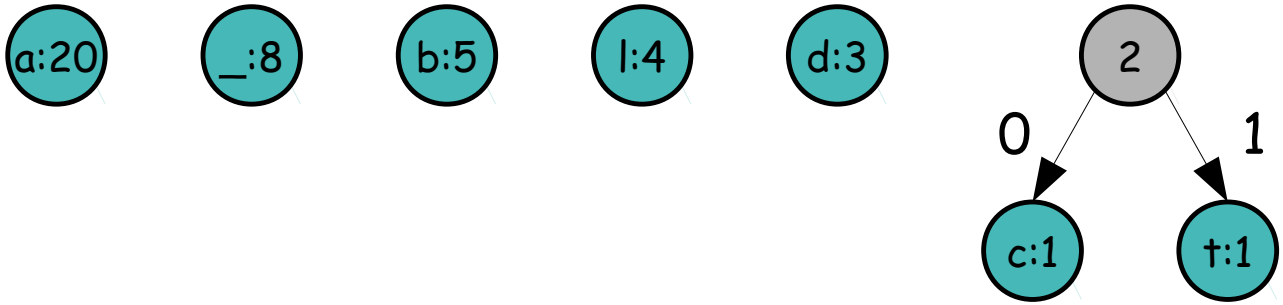
Exemplo

Exemplo

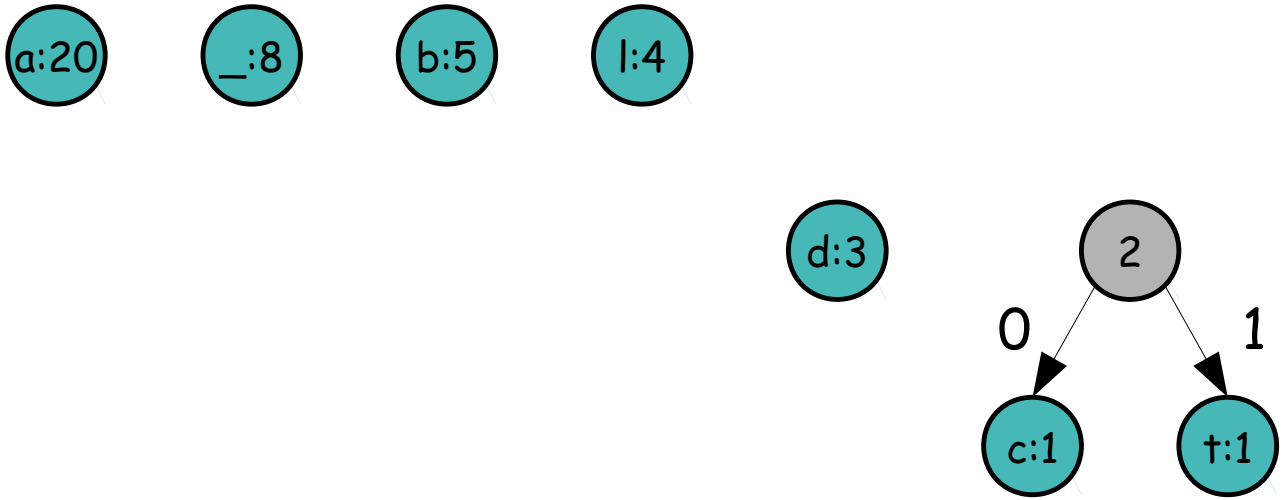
Exemplo



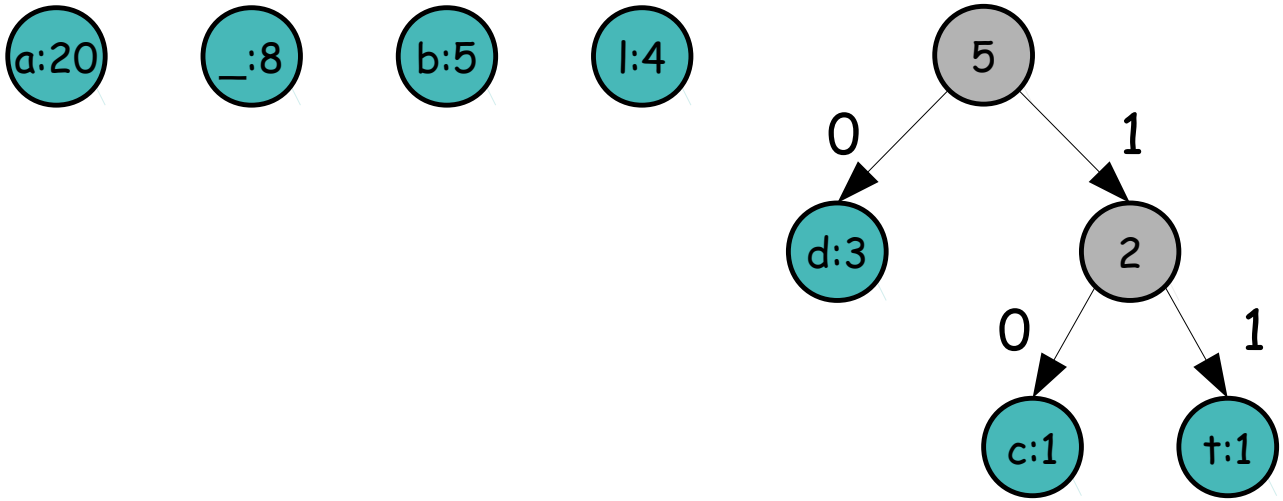
Exemplo



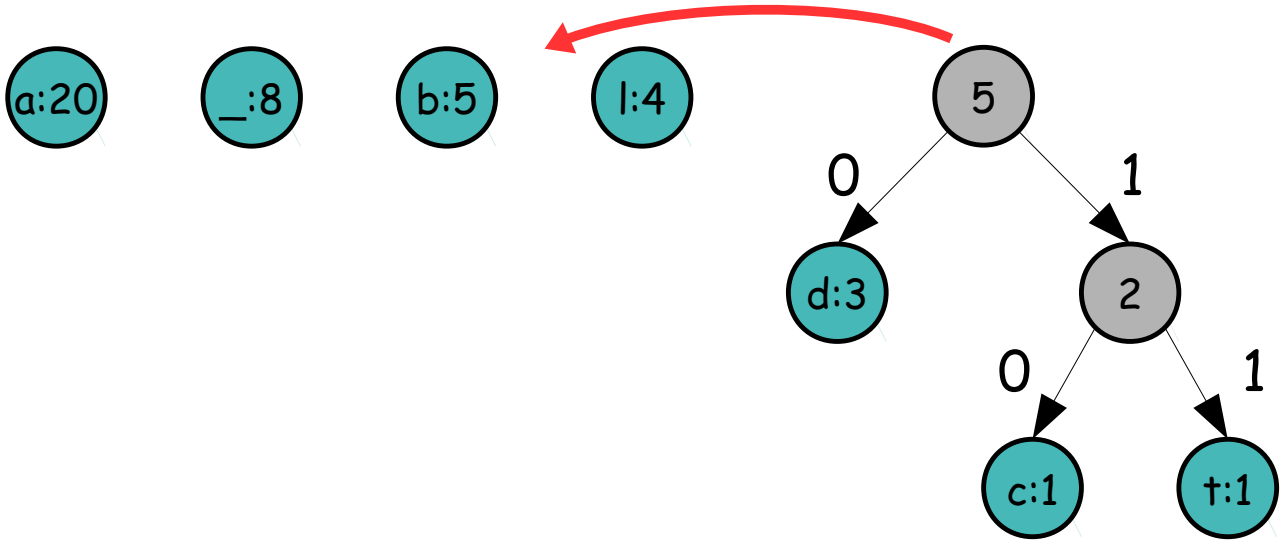
Exemplo



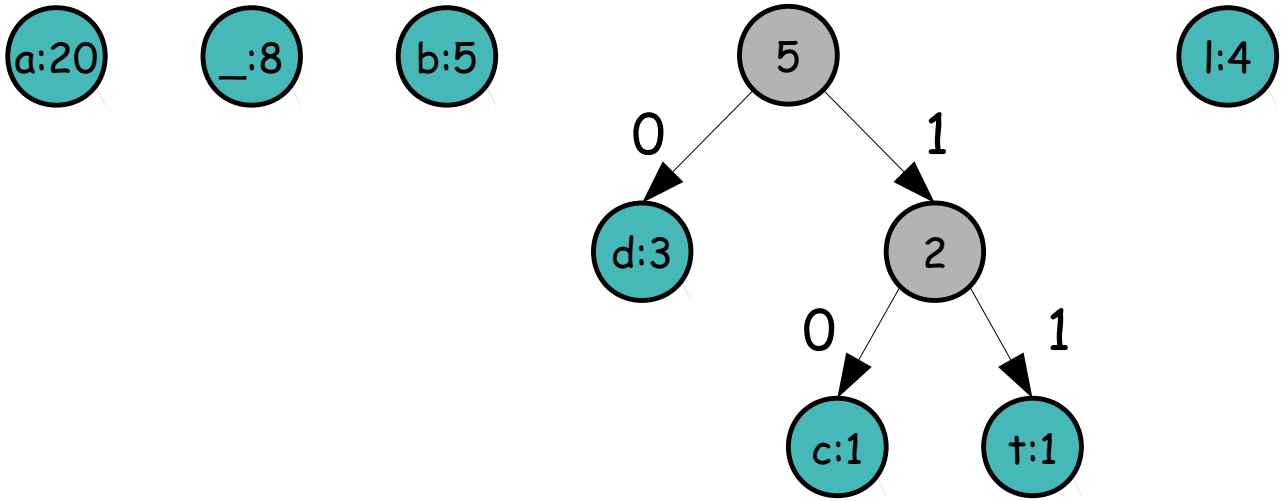
Exemplo



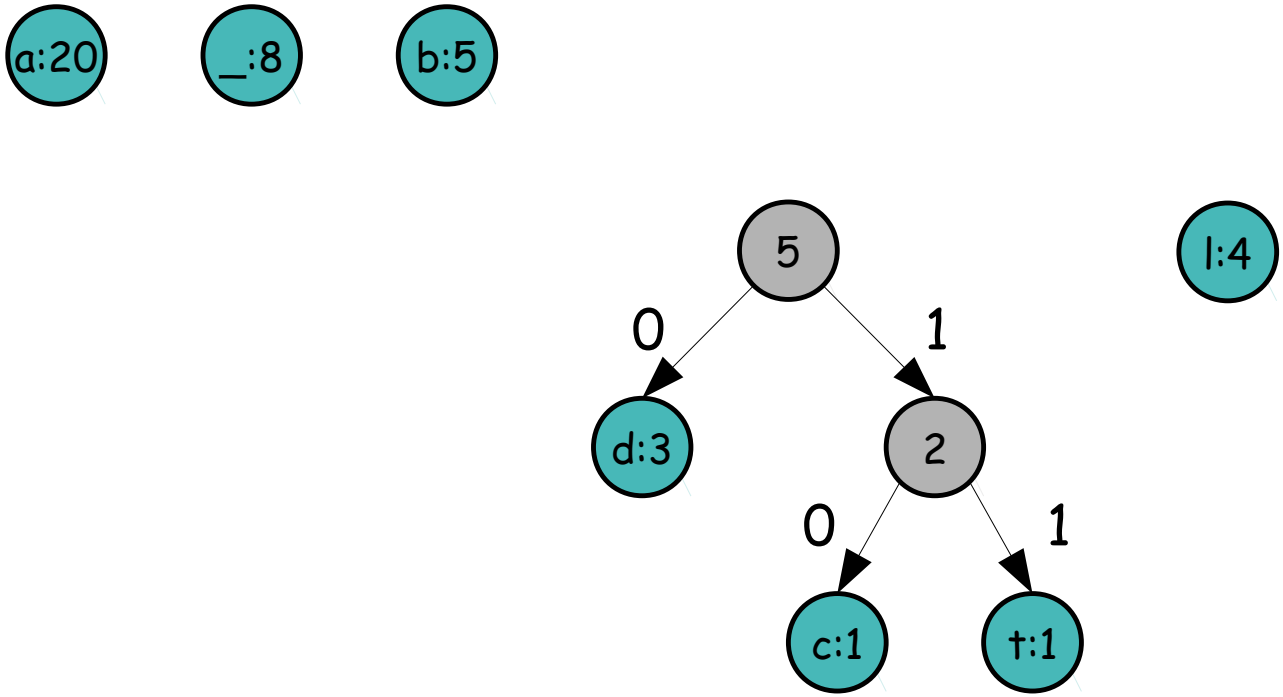
Exemplo



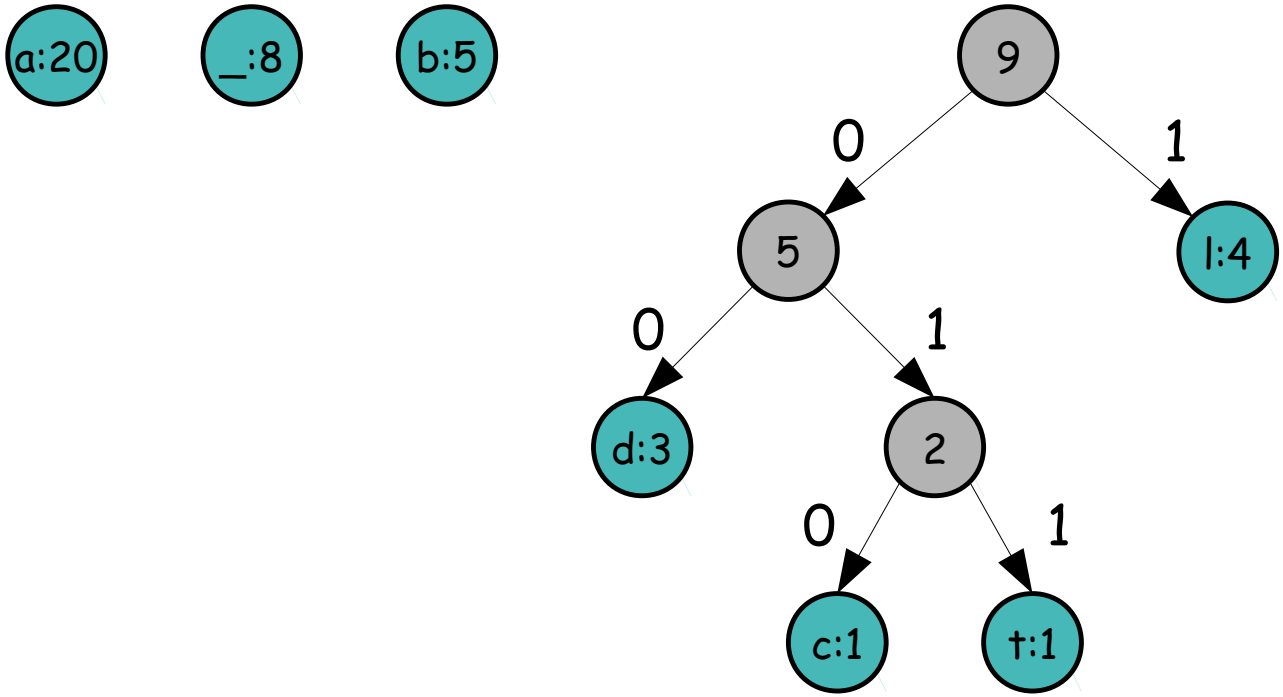
Exemplo



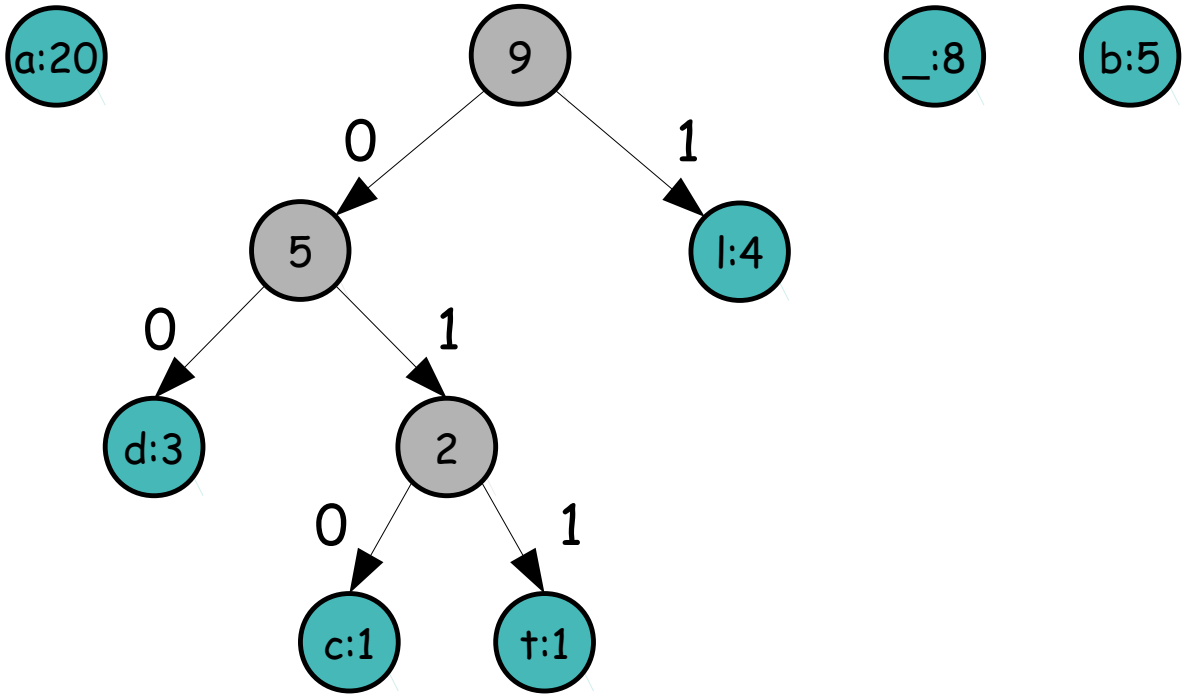
Exemplo



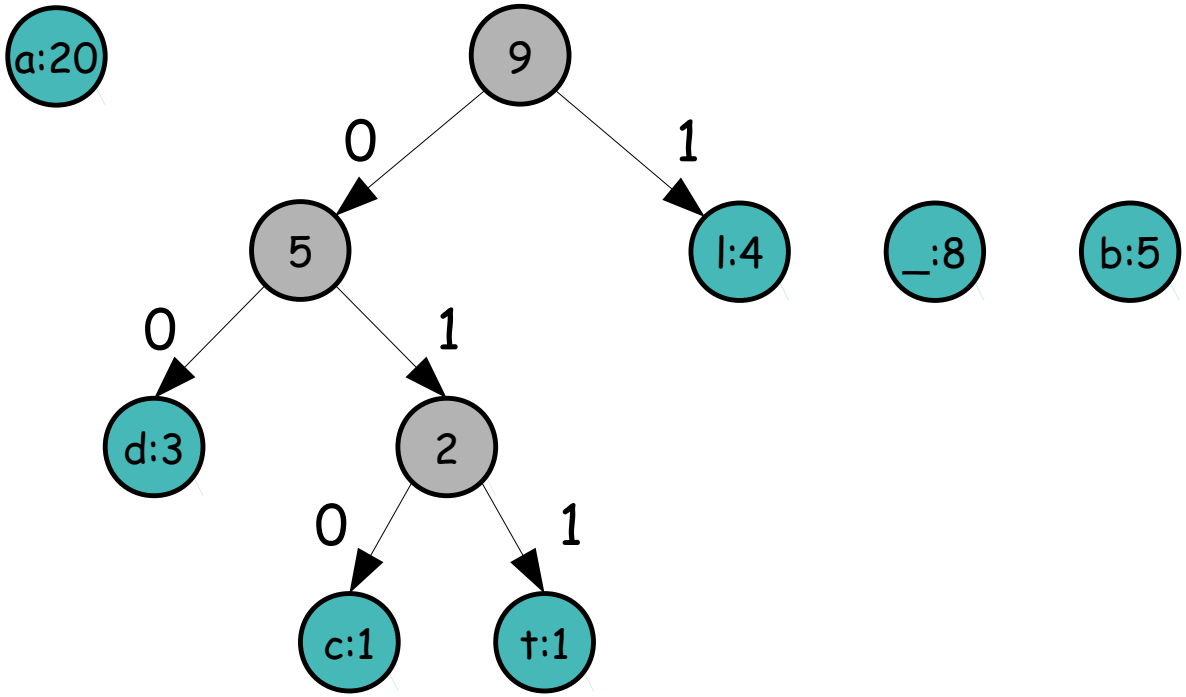
Exemplo



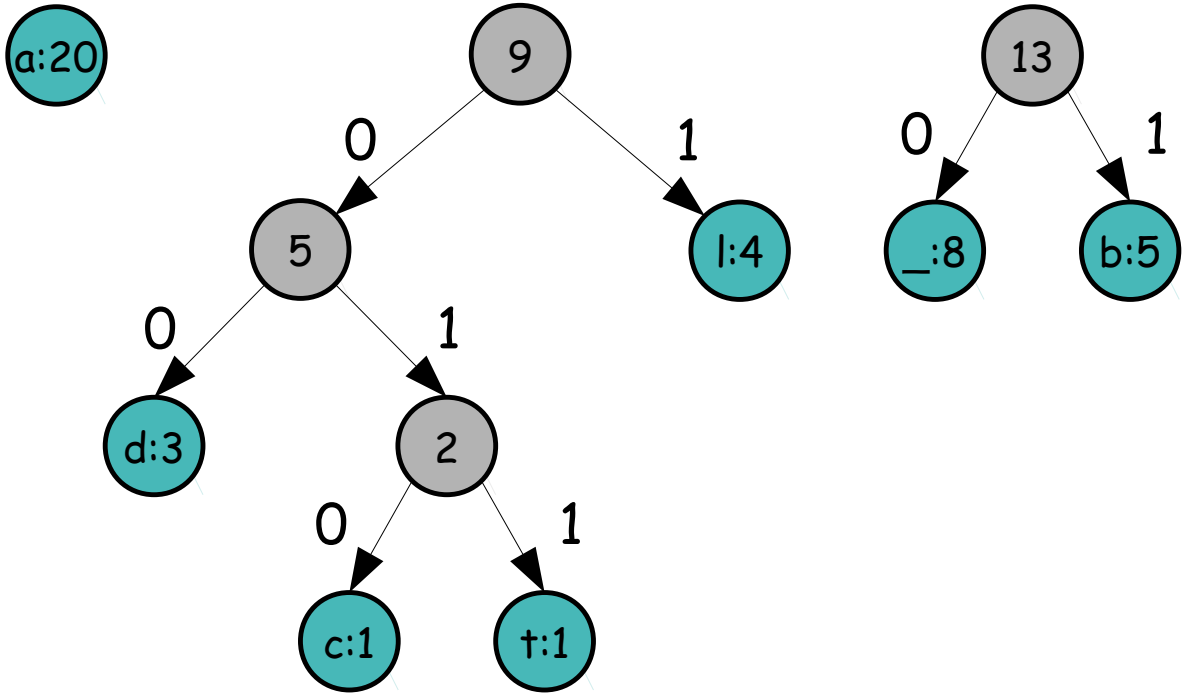
Exemplo



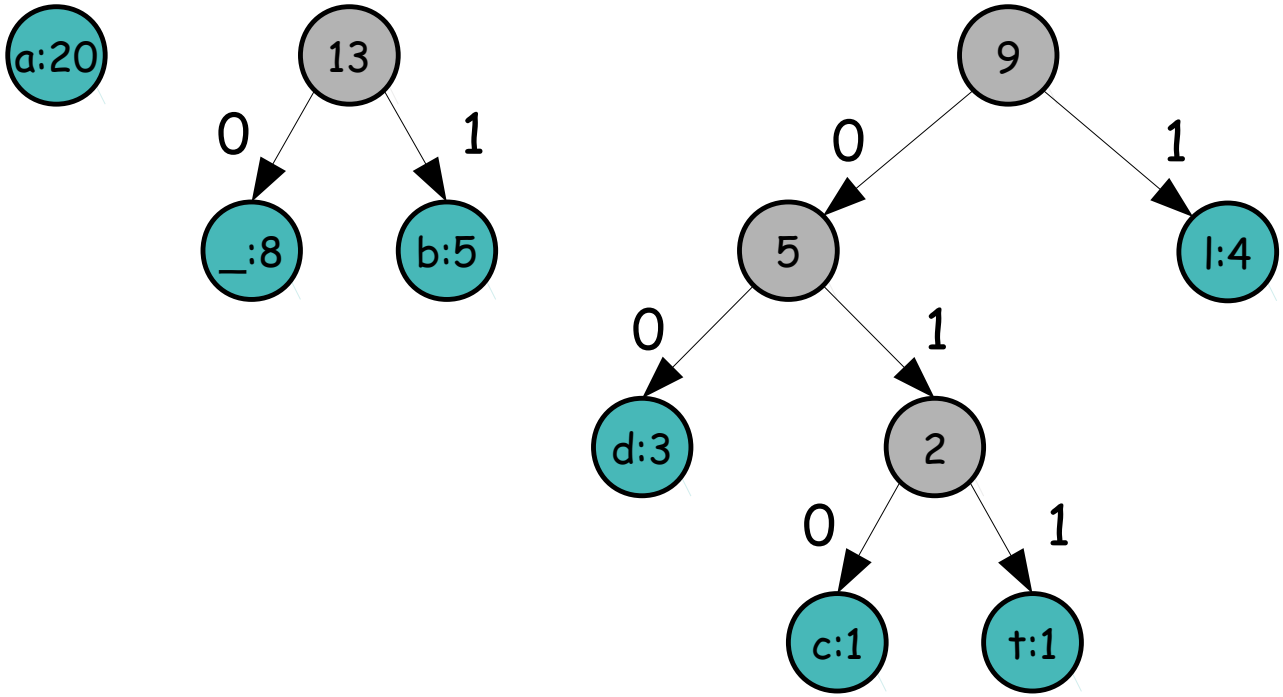
Exemplo



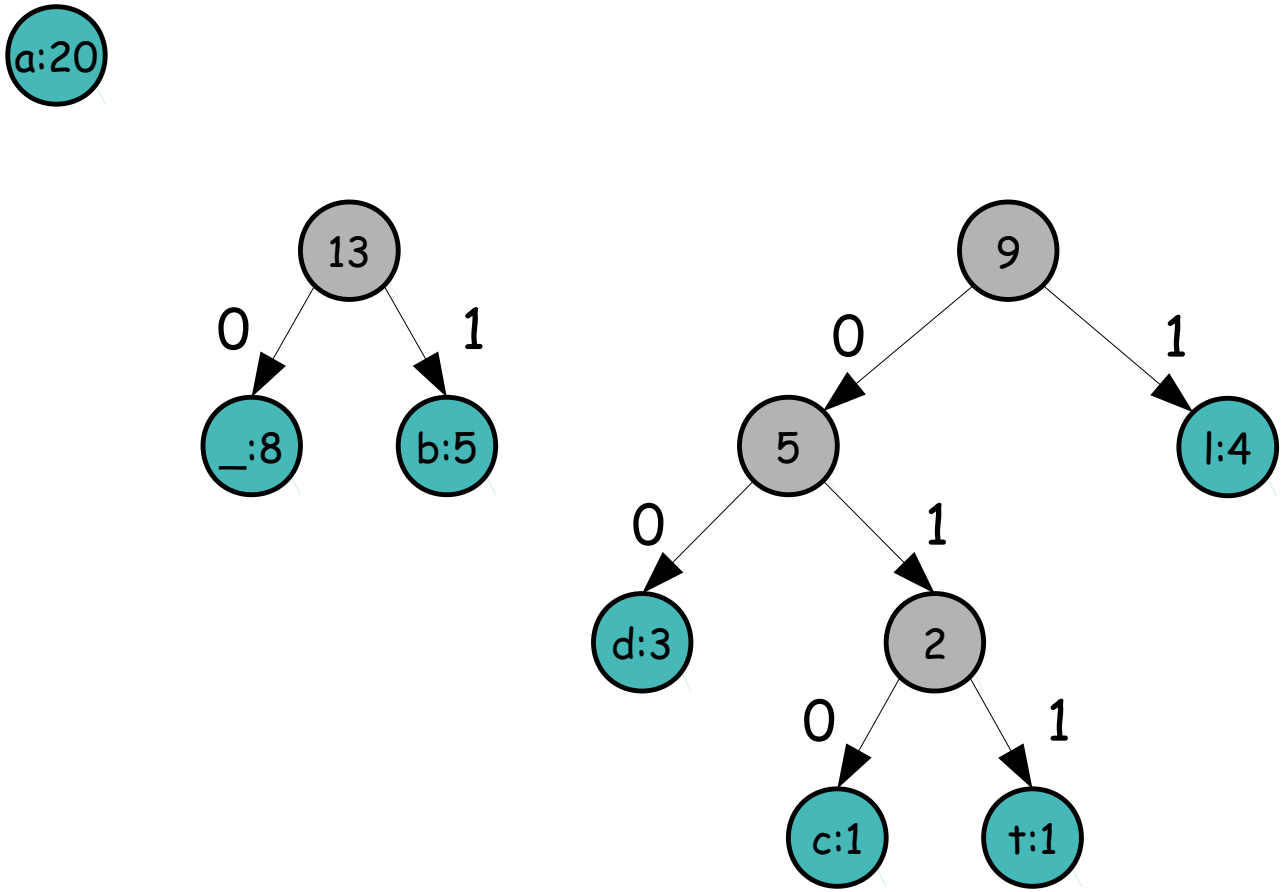
Exemplo



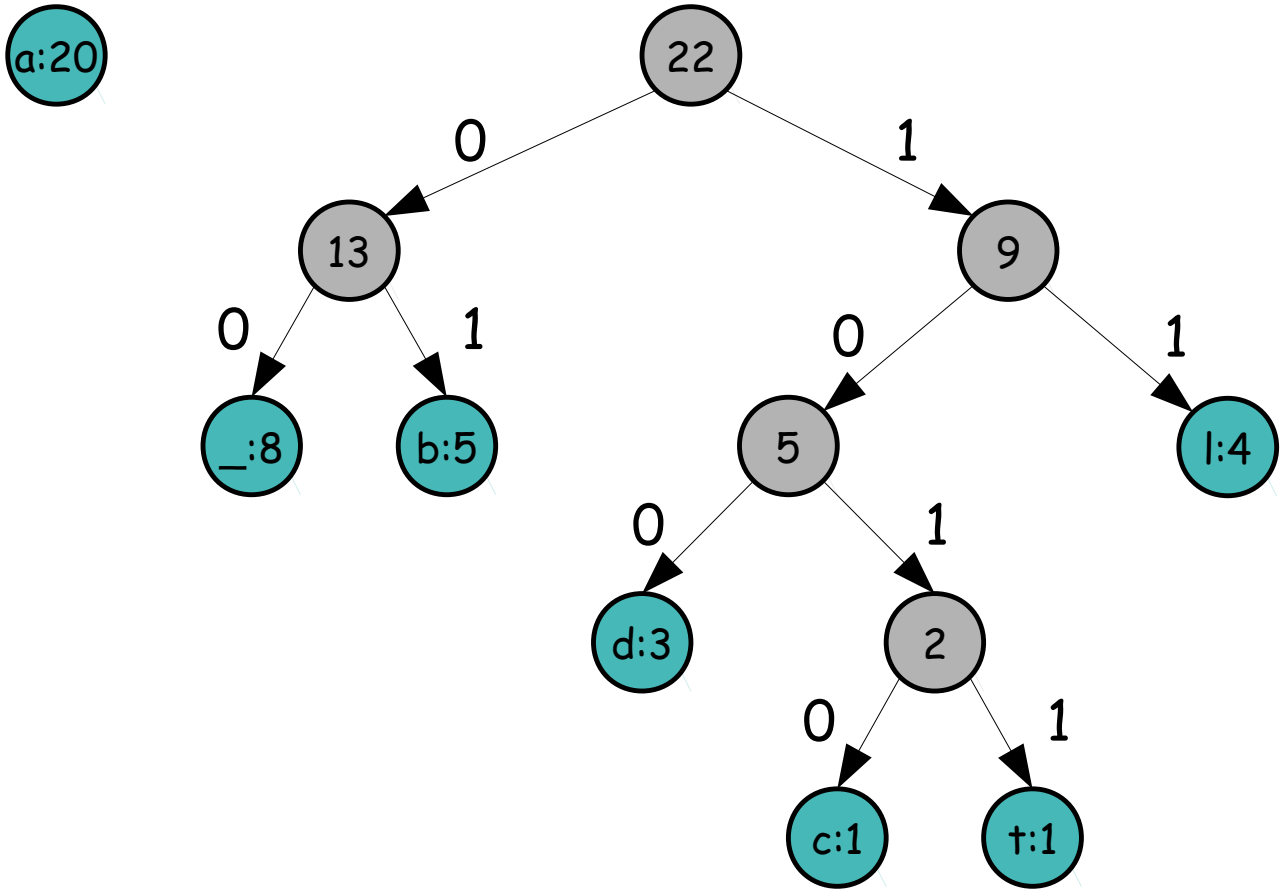
Exemplo



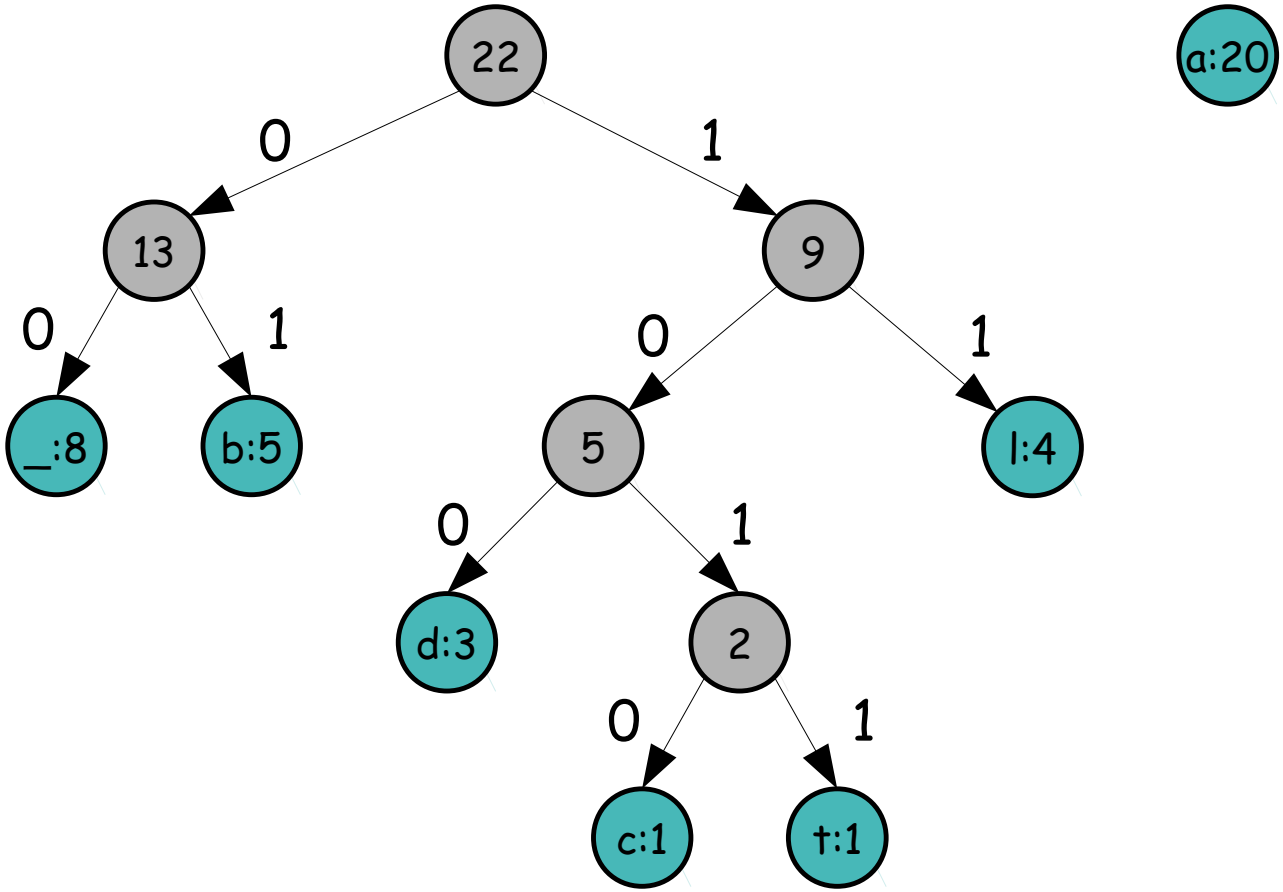
Exemplo



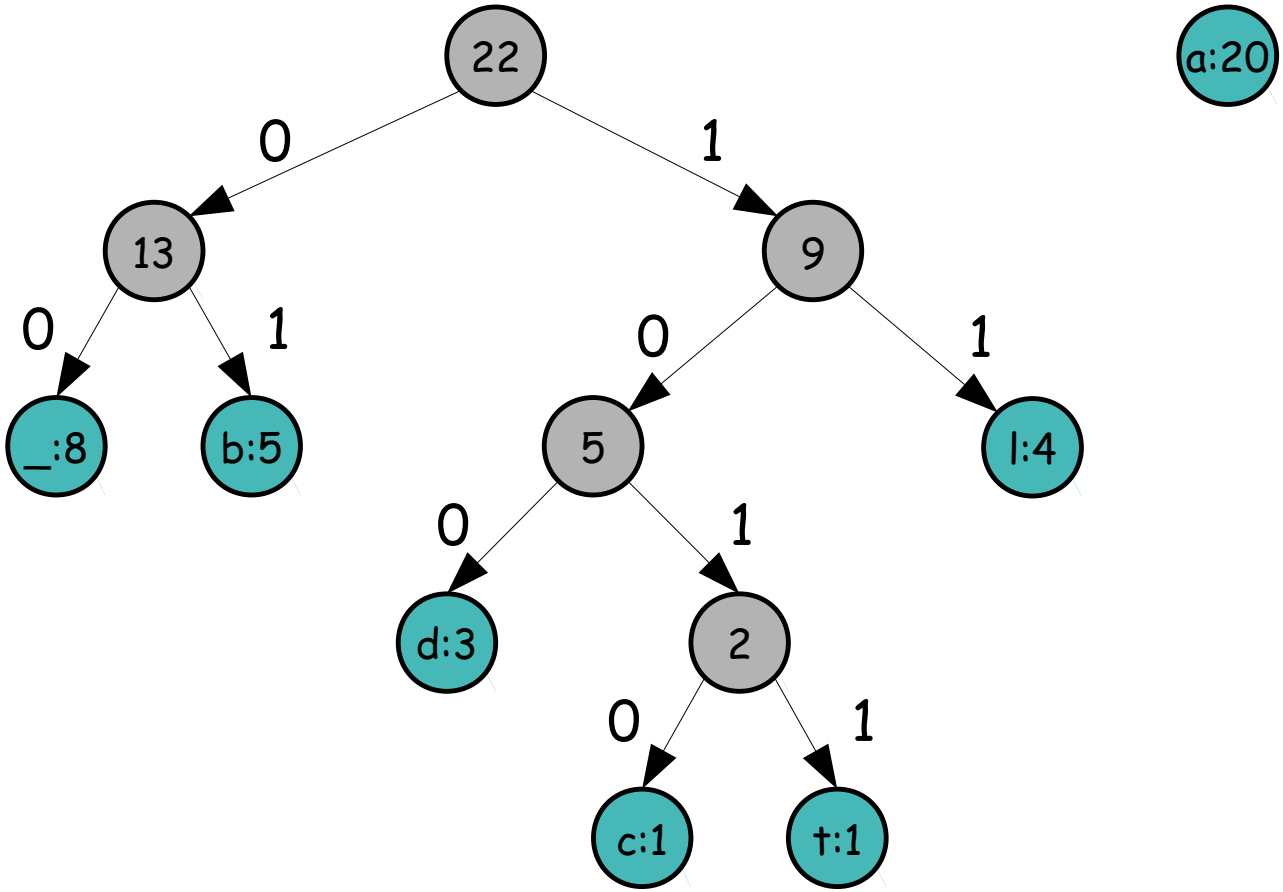
Exemplo



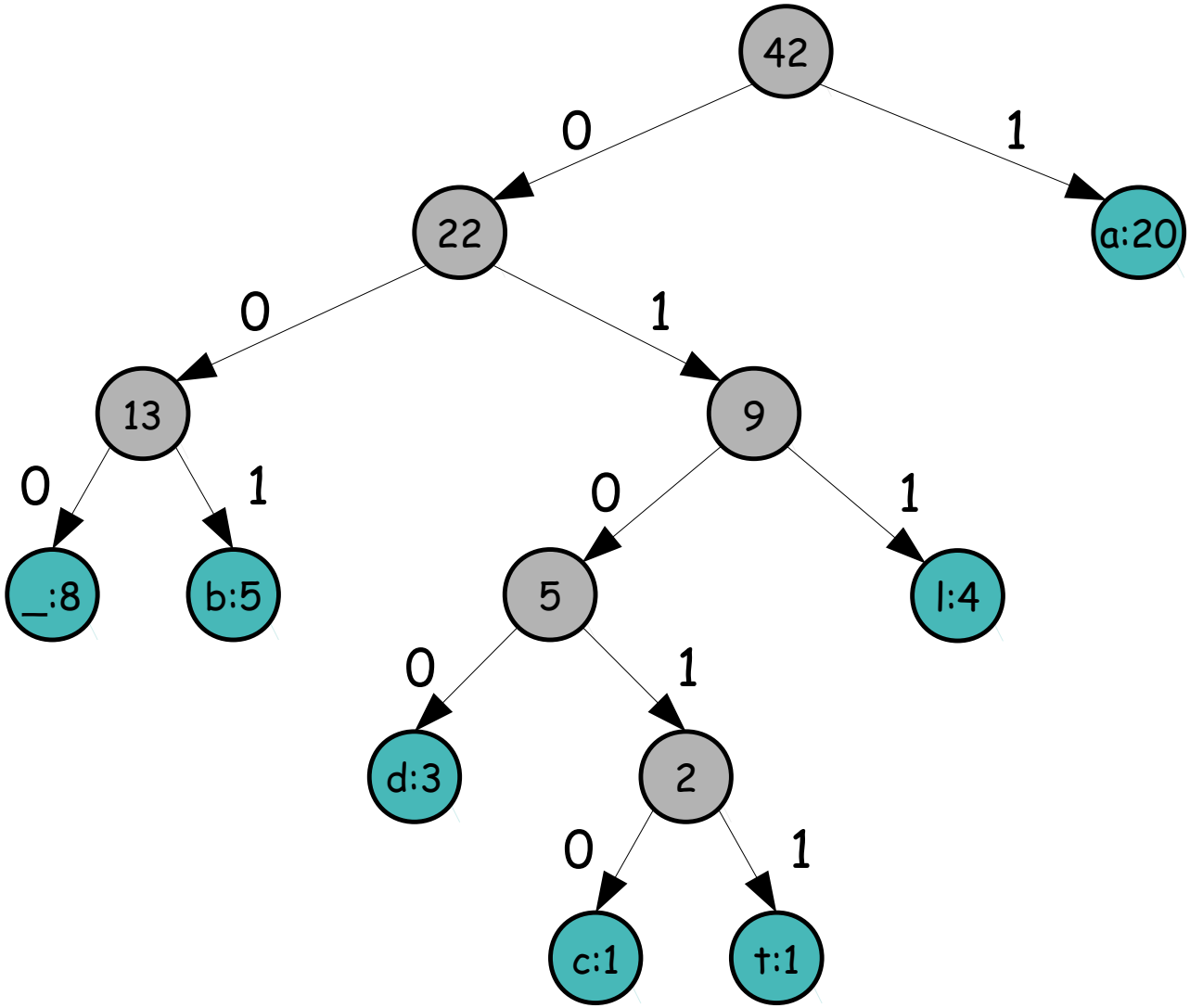
Exemplo



Exemplo



Exemplo



Exercício

1. Após a inserção dos elementos 41, 38, 31, 12, 19 e 8 em uma árvore rubro negra faça a remoção dos elementos 19 e 8.
2. Pesquise o que é uma radix-tree. Defina-a. Crie uma radix-tree para o conjunto de palavras: carapuça, caravela, caramelo, carambola, caridade, caridoso
3. Escreva um algoritmo para criar a tabela de codificação a partir da árvore de Huffman